UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Open Digital Twins for Energy-efficient Smart Buildings

Albert Ferraté Cuartero

Bachelor Thesis Submitted to the Faculty of the
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya

Academic Supervisor: Prof. Dr.-Ing. Sergio Lucia Gil
Technische Universität Berlin

Academic Supervisor: Nuria Duffo Úbeda
Universitat Politècnica de Catalunya

In partial fulfilment of the requirements for the degree in
Telecommunications Technologies and Services Engineering

UPC · ETSETB

Barcelona, June 2020

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

telecos BCN

Acknowledgements

I would like to express my very great appreciation to all the people who supported me during the development of the project, both academically and personally. Many have given me a hand along the way, but there are a certain few without which this work would have never existed.

Firstly, I would like to thank Prof. Dr.-Ing. Sergio Lucia and the Laboratory of Internet of Things for Smart Buildings for hosting and providing me with insight and expertise, which enabled me to completely focus on my work.

Secondly, I am deeply grateful for the assistance given by Nuria Duffo for always answering my doubts, helping me without hesitation and for his patient guidance.

Finally, I would like to thank my family and friends for their unconditional love, who have always supported me on all the important decisions I have made.

Albert Ferraté Cuartero
Berlin, June 2020

Abstract

Energy usage in buildings represents the 40 % of the total energy consumption in Europe, and it is responsible for the 36 % of the CO2 emissions. Smart control systems for the energy management are currently being considered as a promising solution for this problem. These systems rely on the incorporation of a *digital twin* from each building.

Digital twins serve to evaluate the performance of the thermal energy managing system, which are automatically modelled with the use of open data obtained from the German government. Supplementary data such as the construction materials or the weather forecast can be also incorporated, and then a mathematical model which can predict the energy consumption is generated.

Considering that the models derived from the usage of open data are only approximate, this thesis exposes the application of a set of temperature, humidity and light sensors that communicate through a low-power wide-area network called *LoRa* (Long Range), so as to perform the necessary adjustments to the previously calculated models to achieve the individualization required for the energy optimization.

Resum

L'ús d'energia en els edificis representa el 40% del consum total d'energia a Europa, i és responsable del 36% de les emissions de CO2. Sistemes de control intel·ligents per a la gestió energètica estan sent actualment considerats com una solució molt prometedora per a aquest problema. Aquests sistemes consisteixen en la incorporació de *digital twins* per a cada edifici.

Els *digitial twins* serveixen per avaluar el rendiment del sistema de gestió d'energia tèrmica, i son automàticament modelats a partir d'*open data* proporcionada pel govern alemany. Informació addicional com els materials de construcció o les prediccions meteorològiques son també afegides, i posteriorment es genera un model matemàtic que prediu el consum energètic.

Considerant que els models derivats de l'ús d'*open data* son només aproximats, aquesta tesis exposa l'aplicació d'un conjunt de sensors de temperatura, humitat i llum que es comuniquen mitjançant una xarxa de baixa potència i llarg abast denominada *LoRa*, per tal d'efectuar els ajustaments necessaris als models prèviament calculats, i així assolir la individualització necessària per l'optimització energètica.

Resumen

El uso de energía en los edificios representa el 40 % del consumo total de energía en Europa, y es responsable del 36 % de las emisiones de CO2. Sistemas de control inteligentes para la gestión energética están siendo actualmente considerados como una solución muy prometedora para este problema. Estos sistemas consisten en la incorporación de *digital twins* para cada edificio.

Los *digitial twins* sirven para evaluar el rendimiento del sistema de gestión de energía térmica, y son automáticamente modelados a partir de open data proporcionada por el gobierno alemán. Información adicional como los materiales de construcción o las predicciones meteorológicas son también añadidos, y posteriormente se genera un modelo matemático que predice el consumo energético.

Considerando que los modelos derivados del uso de *open data* son sólo aproximados, esta tesis expone la aplicación de un conjunto de sensores de temperatura, humedad y luz que se comunican mediante una red de baja potencia y largo alcance denominada *LoRa*, para efectuar los ajustes necesarios a los modelos previamente calculados, y así lograr la individualización necesaria para la optimización energética.

Revision history and approval record

Revision	Date	Purpose
0	11/06/2020	Document creation
1	23/06/2020	Document revision
2	24/06/2020	Document revision
3	28/06/2020	Document approval

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Albert Ferraté	albert.ferrate@estudiant.upc.edu
Prof. Dr.-Ing. Sergio Lucia	sergio.lucia@tu-berlin.de
Nuria Duffo Ubeda	duffo@tsc.upc.edu

Written by:		Reviewed and approved by:			
Date	14/06/2020	**Date**	24/06/2020	**Date**	23/06/2020
Name	Albert Ferraté	**Name**	Prof. Dr.-Ing. Sergio Lucia	**Name**	Nuria Duffo
Position	Project Author	**Position**	Academic Supervisor	**Position**	Academic Supervisor

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
BLE	Bluetooth Low Energy
BRCM	Building Resistance-Capacitance Modelling
CSV	Comma Separated Values
EU	European Union
GML	Geography Markup Language
IoT	Internet of Things
JSON	JavaScript Object Notation
LoRa	Long Range
PCB	Printed Circuit Board
SOC	State of Charge
TTN	The Things Network

Chapter 1

Introduction

1.1. Background and Motivation

During the last years, the European Union (EU) has been tackling climate change establishing legislative frameworks to raise the use of renewable energies, reduce emissions and to increase energy efficiency.

The energy usage in the building sector represents the 40% of the total energy consumption in Europe [1]. Therefore, this sector is crucial for achieving the EU's energy and environmental goals. The EU created the *Energy Performance of Buildings Directive* to establish strong long-term renovation strategies and other measures to modernise the buildings sector in light of technological improvements. Additionally, better and more energy efficient buildings not only bring benefits to the environment and society, but also help improve the quality of life of its inhabitants.

While one objective is to achieve high energy efficiency, the comfort level of the occupants also needs to be taken into account. This level is frequently highly correlated to the temperature, air quality and visual comfort. Usually, the comfort room temperature is ranged between 20 °C and 22 °C when the outdoor temperatures are between 10 °C and 16 °C [2, page 12, Figure 5.1]. This means the high comfort objective is often contradictory with the energy consumption one.

To understand the behaviour of the room's temperature and humidity, a crucial aspect for the temperature managing through heaters and air-conditionings, buildings can be modelled as a *digital twin* to predict the behaviour of the real building.

1.2. Starting Point

This thesis was proposed by the *Laboratory of Internet of Things for Smart Buildings* as a continuation of an already existing project. The aim of this work is to validate the precision of a prediction software that relies on open-data to generate a simulation of a building's parameters.

The existing software is programmed mostly in Python and, starting from weather data and the geometrical information from a building or district, is able to generate a simulation that evaluates the power consumption, temperature and other parameters from each of the selected buildings.

In order to compare the simulated and predicted parameters with the actual values, data from a real environment is used. Two sensors are deployed to collect temperature, humidity and light measurements, which are then used to be compared with the results obtained from the simulation.

1.3. Incidences

During the development of this project, a worldwide incidence occurred which resulted in major modifications of the project. This incidence is the COVID-19 pandemic outbreak, which started affecting most European countries around mid-March. For this reason, I had to come back to Barcelona after the first month and adapt the plans for the project into a new environment. The changes were significant, since the project relied in open-data exclusively obtainable for Germany. The original plan also involved the deployment of hardware in public buildings and institutions such as the *Robert Koch Forum* in Berlin, headquarter of the *Einstein Center for Digital Future*, where access to its 3D printers and other services to help with the progress of the project would have been given. Some of the necessary materials were brought to Barcelona to continue with the development of the project.

It is also important to note that in my residence in Barcelona I have a workshop equipped with tools and electrical components, and also a 3D printer. This has proven to be especially useful considering that during the lockdown nobody was allowed to leave its residence.

1.4. Requirements and Specifications

Taking into consideration the above-mentioned incidence, this project was adapted to its new location. The following requirements were established after applying the necessary modifications to the project.

- ◇ A network of sensors will be deployed to periodically collect data.
- ◇ The sensors will communicate via LoRa to send the collected information.
- ◇ The sensors will rely exclusively on their own power supply.
- ◇ The battery life will be maximized.
- ◇ The setup will be scalable and identically replicable at any time.
- ◇ The collected data will be displayable and downloadable from any part of the world.
- ◇ The simulation software will be adapted to use Barcelona's weather data.
- ◇ The simulation software will be modified to maximize the similarities between the simulated and real data.

In order to achieve these requirements, the following specifications must be met.

- ◇ Two sensors will be deployed and will collect and send data every 30 minutes.
- ◇ A LoRa Gateway will be installed near the sensor devices to ensure a robust connection.
- ◇ To eliminate the need of an external supply line, small batteries will be used.
- ◇ Energy harvesting mechanisms based on a solar panel will be implemented to maximize the battery life by at least a 20%
- ◇ Casing for the sensor devices will be 3D printed to ensure a proper standardization for the setup. The 3D designs will be downloadable from any part of the world.
- ◇ A dashboard to visualize the data will be designed and published in a webpage in order to have access to the data from anywhere.

1.5. Time Plan

Figure 1.1 depicts the time management of the project.

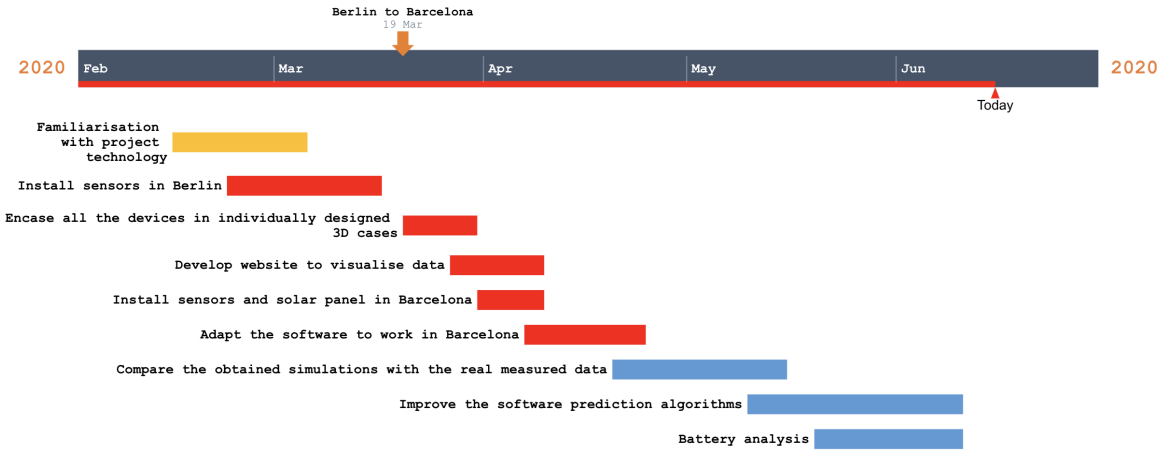


Figure 1.1: Gantt diagram of the project

Chapter 2

State of The Art

This thesis is part of a larger project started in the Laboratory of Internet of Things for Smart Buildings, and this chapter gives a brief introduction on the state of the technologies and software involved in the project. It is not meant to be a comprehensive summary of the theory but rather a short insight.

2.1. Sensor Setup

A UPC master student, Marcos Torres Padín, developed an energy efficient software to be installed in the sensor devices in his master thesis [3]. A brief description about his work will be done, which can be divided in three steps: measurement, communication and collection.

Measurement

The sensor data is measured using the Pysense module from the manufacturer Pycom. This board contains various useful features, such as temperature, light, barometric pressure and humidity sensors. An additional temperature sensor is also installed with an extension cable in order to measure the temperature of a near heater.

The temperature, light, pressure, humidity and battery voltage are being measured every 30 minutes, and then the data is encoded in an array of bytes ready to be sent.

Communication

The Pysense module has attached a LoPy4 device, which is developed by the same manufacturer. This device is equipped with WiFi, LoRa, Bluetooth Low Energy (BLE) and Sigfox[1] modules. To send the data, a third party, open-source and decentralized network is used. This network is called *The Things Network* (TTN) and it enables the device to use long range gateways for low power devices, especially useful for Internet of Things[2] (IoT) applications.

Gateways form the bridge between devices and The Things Network. Nodes use low power networks like LoRaWAN to send messages to the gateway, whereas the gateway uses high bandwidth networks like WiFi, Ethernet or Cellular to connect to The Things Network. All gateways within reach of a device will receive its messages and forward them to TTN, which has over 10.000 gateways in 150 countries, and most of them are public and can serve several devices.

Collection

Once the encoded data is received by the gateway, it is forwarded via WiFi to The Things Network, where it is decoded. An Angular interface is used to visualize the data gathered by the sensors as well as the Berlin weather data. The overall view of the setup is depicted in Figure 2.1.

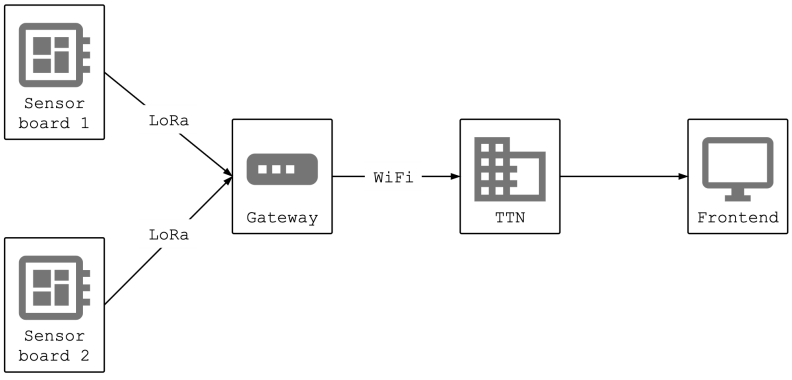


Figure 2.1: Overall view of TTN setup.

[1]Sigfox enables communication using a radio band with a central frequency of 868MHz in Europe. It utilizes a wide-reaching signal that requires little energy, and it is often used to build wireless networks to connect low-power objects.

[2]The *Internet of Things* (IoT) is a system of interrelated computing devices that have the ability to transfer data over a network without requiring human interaction.

2.2. Prediction Software

The Laboratory of Internet of Things for Smart Buildings from the TU developed a software that, starting from weather data and geometrical information from a building or district, is able to generate a simulation analysing the power consumption, temperature and other parameters from each of the buildings. The resulting software diagram is presented in Figure 2.2. More information about the software can be found in [4, 5]. The four different packages that compose it are summarized hereafter.

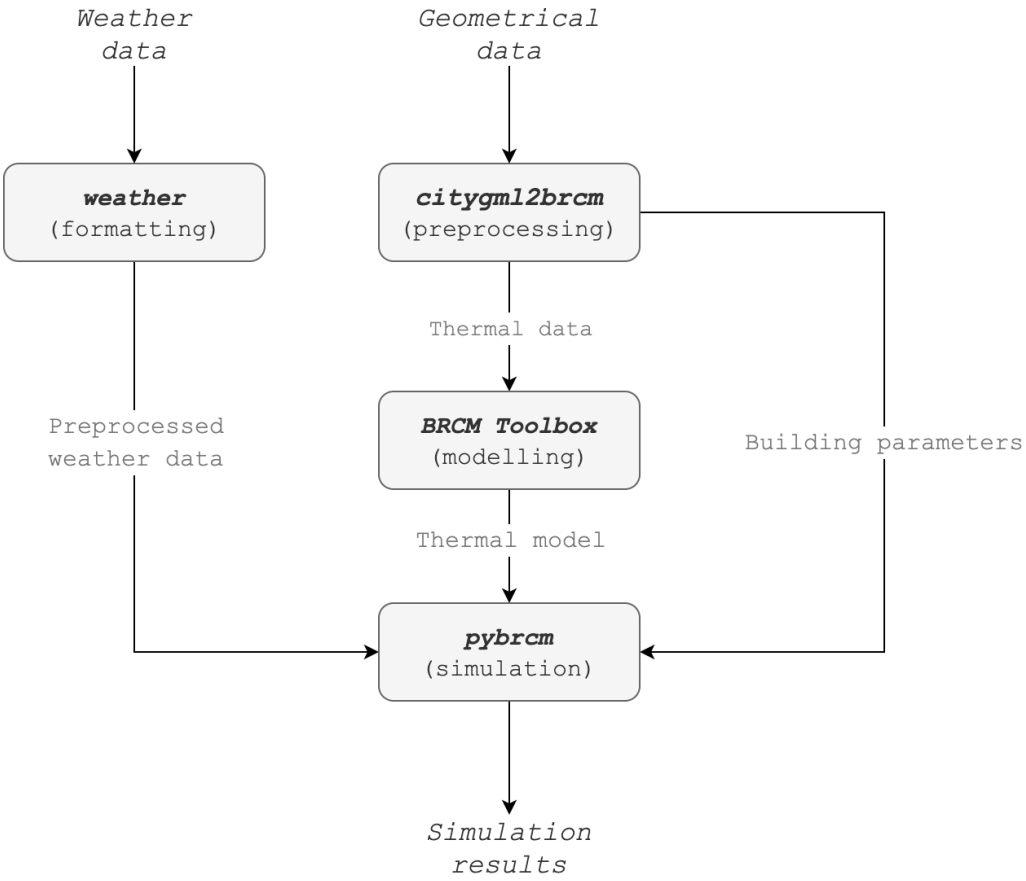


Figure 2.2: Simplified diagram of the simulation software.

The first one is called *weather*, and it has a simple purpose: create a *Comma Separated Values*[1] (`*.csv`) file containing all the weather information in the specific format required for the simulation. The input files must contain hourly data of the ambient temperature, soil temperature and insolation time.

The second package, *citygml2brcm*, takes as an input a *CityGML* file (`*.gml`), which

[1] A Comma Separated Values (CSV) file is a text file that uses a delimiter to separate values. Each line of the file is a data record, which consists of one or more fields, separated by delimiters.

is an open standardised data model to store digital 3D models of cities implemented as a GML application[1]. It contains geometrical information about the building or district under analysis. The software performs a preprocessing to determinate and define each individual building of the selected area. It then creates 10 `*.csv` files containing a thermal model and an external heat flux model. It also generates a `*.pickle`[2] file with other relevant building data.

The package *brcmtoolbox* is implemented in MATLAB. It facilitates the *Building Resistance-Capacitance Modelling* (BRCM) using the files provided by *citygml2brcm*. It outputs a `*.mat`[3] file containing matrices with the thermal district model.

The last package, *pybrcm*, requires the properly preprocessed weather data (`weather.csv`), the building parameters created by *citygml2brcm* (`buildings.pickle`) and the thermal district model (`model.mat`). Using these parameters, a simulation is generated which outputs information such as the temperature estimation and energy usage for each of the buildings.

[1]The *Geography Markup Language* (GML) is a markup language that expresses geographical features. GML serves as a modeling language for geographic systems as well as an open interchange format for geographic transactions on the Internet.

[2]The pickle module, which is supported by Python, implements binary protocols for serializing and deserializing a Python object structure. *Pickling* is the process whereby a Python object hierarchy is converted into a byte stream, and *unpickling* is the inverse operation, whereby a byte stream is converted back into an object hierarchy.

[3]Files with the .mat extension are files that are in the binary data container format that the MATLAB program uses. The extension is categorized as a data file that include variables, functions, arrays and other information.

Chapter 3

Implementation

3.1. Sensor Deployment

Since the system is required to be scalable and replicable, a proper standardization needs to be performed. The standardization of the measurement setup guarantees that the same exact procedure can be replicated in another part of the world at any point.

Additionally, the existing sensor setup had exposed connections, being the external temperature sensor especially affected. To prevent a bad electrical contact and therefore unreliable measurements, a protective case for each of the devices is needed. Using the *Autodesk Fusion360* software, a case is designed and printed using a 3D printer. The case is sketched to intentionally leave the sensors exposed, therefore ensuring that the measurements from the light sensor are not being disturbed. Figure 3.1 depicts the designed model, and a picture of the printed result can be seen in Figure A.1.

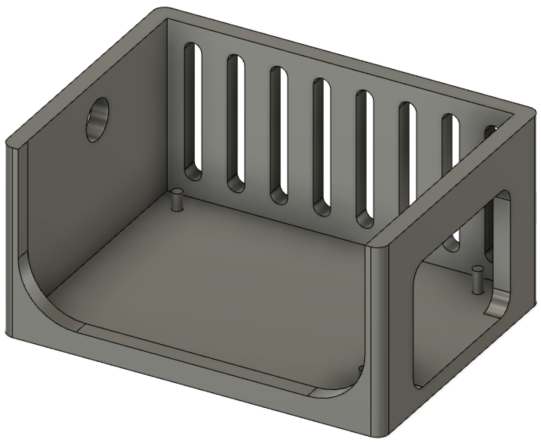


Figure 3.1: Sketch of the sensor board protective case.

A smaller case is also designed for the external temperature sensor board, which is presented in Figure 3.2. This case is particularly important considering the high temperatures a heater can reach. Pictures of the printed result can be seen in Figure A.2.

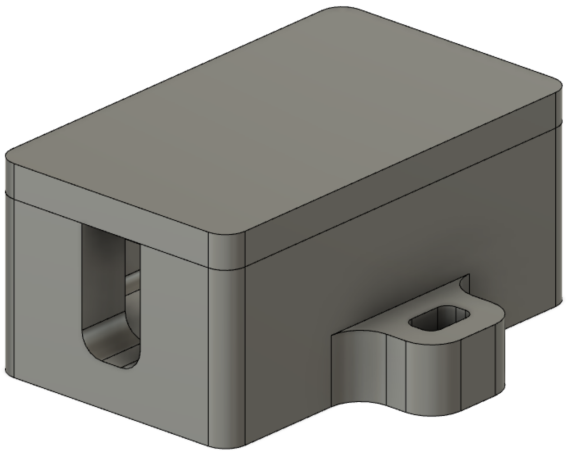


Figure 3.2: Sketch of the external sensor protective case.

Once the sensors are encased, the setup is prepared to be deployed. Both devices are installed in different rooms from the same residence. In Appendix A.1 pictures of the installed setup can be found.

3.2. Website Development

Initially, the data was being visualized using an Angular application which displayed both the measured data and Berlin's weather data. When the project had to be adapted from Berlin to Barcelona, this API was no longer useful since it did not have available weather data from Barcelona. Additionally, in order to adapt the project to its new context, it is preferred to use a public webpage so that all the project members can visualize the data taken with the sensors from any part of the world. See Appendix A.2 for screenshots of the webapge.

The website enables the user to select any of the two sensor devices, and then visualize specific sensor data in the specified range of dates. The website can be accessed through the following URL or using the QR code in Figure 3.3 (only compatible with desktop devices, not with mobile devices).

`http://ec2-34-228-44-224.compute-1.amazonaws.com/`

Figure 3.3: QR linking to the designed website.

Figure 3.4 depicts a block diagram for the website execution workflow. It is created using *Django*, an open-source framework for web development, and hosted by *Amazon Web Services*.

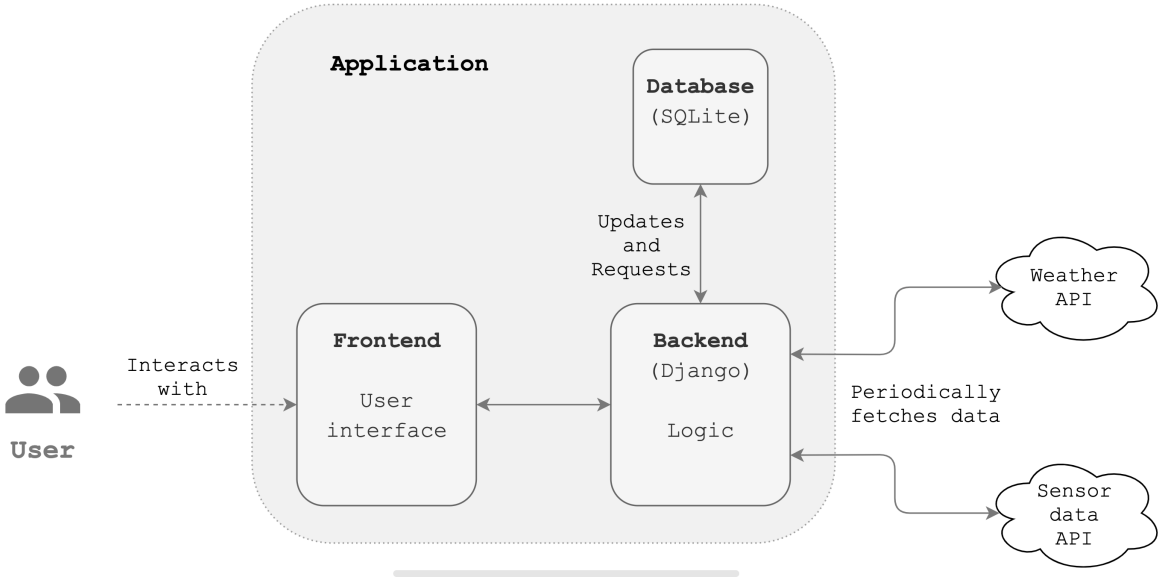


Figure 3.4: Website execution workflow.

Considering that The Things Network API framework [6] to extract the sensor's data only stores information for 7 days, the webpage needs to have its own data base in order to be able to access data from longer periods of time. Consequently, an always-running script that periodically requests the data is deployed in the server and continuously updates the database.

The website also displays the weather forecast from Barcelona to compare the indoor measurements with the external values. Since the chosen weather API [7] only provides the hourly weather information from the last 24 hours, another always-running script that requests weather data and uploads it to the database is implemented to remove

this limitation.

In order to facilitate future data processing, a feature to download the raw data from the selected device and data range is also implemented. JavaScript Object Notation (JSON)[1] is the chosen extraction format to simplify the data parsing.

3.3. Energy Harvesting

Battery optimization in IoT networks is a crucial aspect, especially considering that an IoT device is essentially worthless if its power supply is exhausted or unreliable. On account of the needs of this project, where the sensor devices could be installed in a remote location, therefore probably being inaccessible for a long period of time, it is important to focus efforts into prolonging the battery life as much as possible.

There are different approaches in which battery life can be prolonged, but the first and most important one is to optimize the energy used as much as possible. Marcos Torres already tackled this topic in his Master thesis by implementing a mechanism in which the devices go into sleep mode once they send the data to the nearest Gateway.

As a secondary approach to prolong the battery life, an energy harvesting mechanism is also implemented. Marcos did some testing with a solar panel but a proper implementation of the system was never evaluated for a long period of time. Figure 3.5 depicts the setup that is installed in one of the devices. A solar charger is required to charge the battery using solar power.

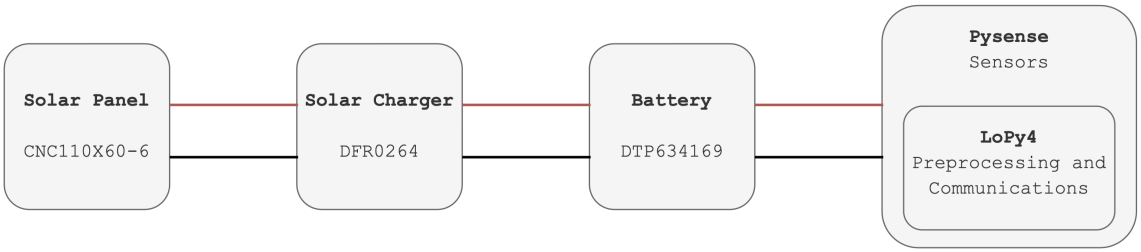


Figure 3.5: Diagram of the solar panel setup.

A support for the solar panel, which also encases both the battery and solar charger, is designed using Autodesk Fusion 360 and printed using a 3D printer (Figure 3.6). It also includes a rotatory mechanism to manually orientate the panel to the desired

[1] JavaScript Object Notation (JSON) is an open standard file format, and language independent format, that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and array data types.

direction. Pictures of the printed supporting structure can be found in Appendix A.3.

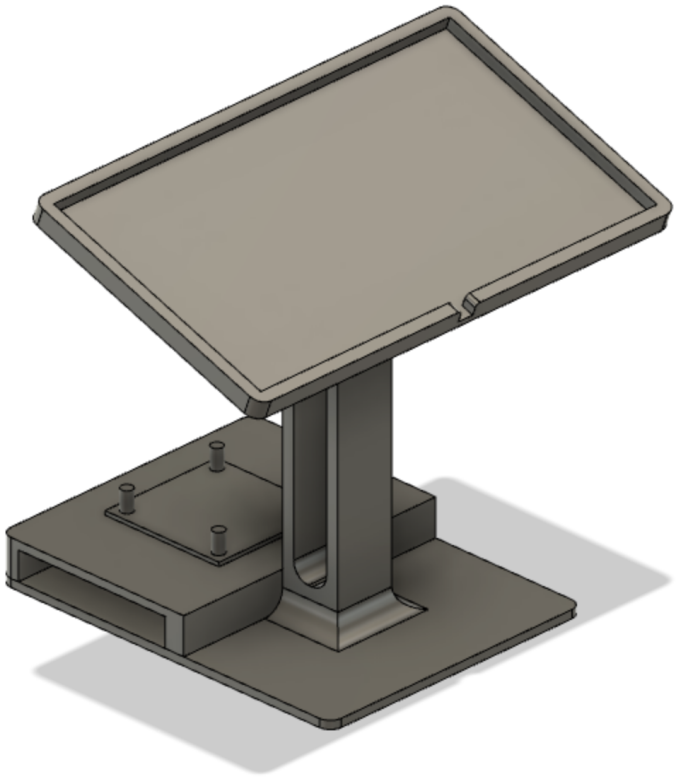


Figure 3.6: Sketch of the solar panel case.

3.4. Software Adaptation

As previously mentioned, this project is started in Berlin, and, around mid-March, it is adapted to Barcelona. This implies many modifications must be made, mostly because the simulation software heavily relies in Berlin data and specifications.

The first and most notable adaptation is made to the *weather* package. Originally, it obtained the data from a Germany's government website [8], which provides hourly measurements of air temperature, ground temperature and insolation minutes[1] from Berlin. This data is given in three different `*.csv` files with a specific format and in German language, so the *weather* package is designed to be able to read these files.

To fetch the hourly weather data from Barcelona, a Spanish website containing the three required parameters is selected[2]. Nevertheless, its output format does naturally not match the required one. Instead of three individual `*.csv` files for each parameter,

[1] *Insolation minutes*: Number of minutes within an hour during which there is sun exposure.
[2] The same webpage that is used in Section 3.2 to get Barcelona's weather forecast.

only a single `*.json` file is given with all the information. Consequently, a Python script is created to convert the `*.json` file into `*.csv` with the required format in order to be readable by the existing software (see script in Appendix B.5).

Once this script is created and the required `*.csv` file is generated, the *weather_preprocess.py* script can be executed for the specified dates, which outputs a `*.csv` file preparing the weather data for the following steps.

On top of that, and as it has been explained in Section 3.2, the Spanish weather API only has downloadable data from the last 24 hours. To avoid manually downloading the data each day, a Python script is created that periodically updates a local `*.json` file containing all weather data from Barcelona (see Appendix B.3).

Chapter 4

Sensor Data Evaluation

From the previous chapter it is known how to acquire and visualize data from any of the devices, which can be regarded as a crucial step to progress in this project. This chapter aims to perform an analysis on the influence that the solar panel has on the battery life (Section 4.1), to evaluate the accuracy of the existing simulation software (Section 4.3) and to tweak its algorithms ensuring that the simulation is as close as possible to the reality (Section 4.4).

4.1. Battery Evaluation

4.1.1. Battery Voltage Analysis

In order to prove the effectiveness of the solar panel, the battery voltage, which is proportional to the *State of Charge* (SOC)[9, page 4, Figure 1] [10], is continuously measured in one of the devices with and without the solar panel to evaluate its performance. Since the LoPy4 device has the ability to read the battery voltage, the value is encoded and sent with the other sensor measurements.

Figure 4.1 illustrates the battery voltage during the 62 days of measurements (the script used to visualize the data can be found in Appendix B.2). Initially, the solar panel is not installed, but after the initial 11 days the battery is charged back to full (which can be seen as a short voltage pulse to five volts) and the solar panel setup described in the previous section is added to the system.

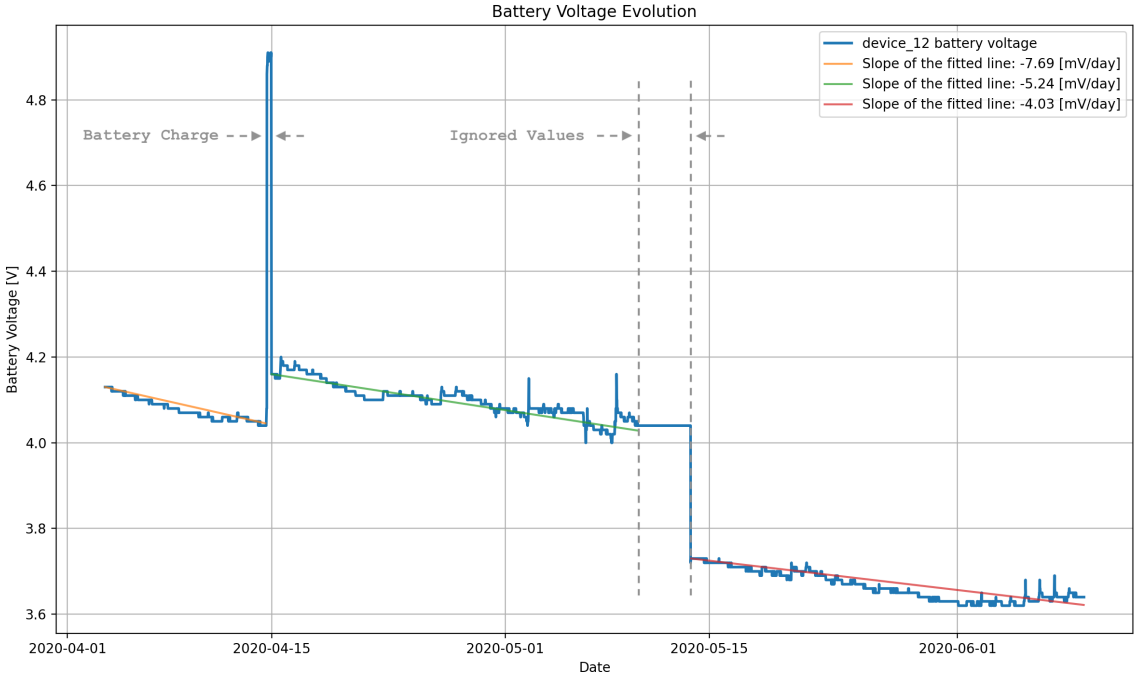


Figure 4.1: Battery voltage measurements.

In the middle of the discharging period the device stopped working for three days. When it was restarted, the battery voltage had decreased from 4.04 V to 3.72 V. Even though the cause of this event is still unknown, it can probably be attributed to a bad connection between the LoPy4 and Pysense modules, since the only way that the device was finally turned back on was by disconnecting and reconnecting both modules. To keep the results as accurate as possible, this event is ignored.

In order to quantitatively analyse the rate at which the battery discharges, the voltage decrease rate is calculated. Hence, three different slopes are calculated: one for the initial discharge without the solar panel (orange fitted line in Figure 4.1), and two slopes for the second discharge with the solar panel, before the voltage jump (green fitted line) and after the voltage jump (red fitted line). Table 4.1 summarizes the obtained discharging rates for each of the time periods.

	No solar panel	With solar panel (1st half)	With solar panel (2nd half)
Number of days	11	25	27
Voltage decrease rate [mV/day]	7.69	5.24	4.03

Table 4.1: Slopes for each time period.

As it can be seen, the voltage decrease rate before the addition of the solar panel (7.69 V/day) is remarkably higher than the decrease rate after its addition (4.61 mV

per day[1]). This means that the addition of the solar panel clearly had a positive effect on the prolongation of the battery life.

Given that the battery has experimentally proven to stop working when the battery voltage drops below approximately 3.4 V, the battery lifespan can be extrapolated from the data assuming a linear discharging behaviour using Equation 4.1.

$$lifespan = \frac{V_f - V_0}{d_r} \tag{4.1}$$

where V_f denotes the voltage of the battery before exhausting (3.4 V), V_0 the initial voltage, d_r the discharging rate and the resulting lifespan is given in days.

For the case without the solar panel, although the sample is small (11 days), the approximate theoretical battery lifespan is calculated using Equation 4.1, which would be of 98 days[2].

On the other hand, repeating the calculations with the solar panel installed and ignoring the voltage jump, the theoretical battery lifespan would be of 164 days[3], which, compared to the previous case, it means an improvement of a 67%. The jump supposed a lifespan loss of 69 days[4] which can naturally not be neglected. Therefore, subtracting these 69 days to the previously calculated lifespan of 164 days, a more accurate value of 95 days can be expected (of which 62 days have already passed).

[1]This value can be obtained by doing the weighted arithmetic mean between the values before and after the voltage jump.
[2]Calculated using $V_f = 4.16$ V, $d_r = 7.69$ mV/day.
[3]Calculated using $V_f = 4.16$ V, $d_r = 4.61$ mV/day.
[4]Calculated using $d_r = 4.61$ mV/day and the voltages before (4.04 V) and after (3.72 V) the voltage jump.

4.1.2. Solar Panel Current Analysis

In contemplation of the results obtained, the generated current by the solar panel is measured and monitored to further analyse its effectiveness. Figure 4.2 depicts the setup used to do the measurements.

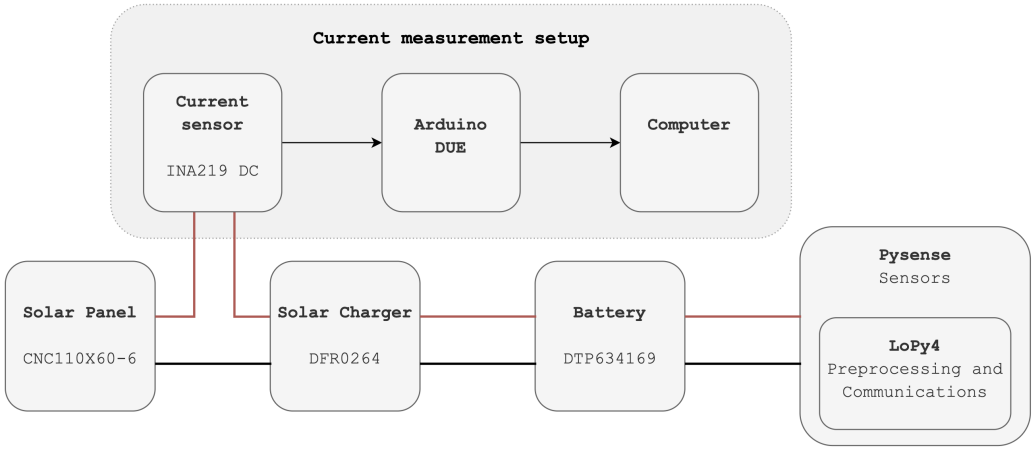


Figure 4.2: Setup used to measure the current.

The generated current by the solar panel and the measured light values are plotted in Figure 4.3. As expected, a high correlation between both parameters can be appreciated.

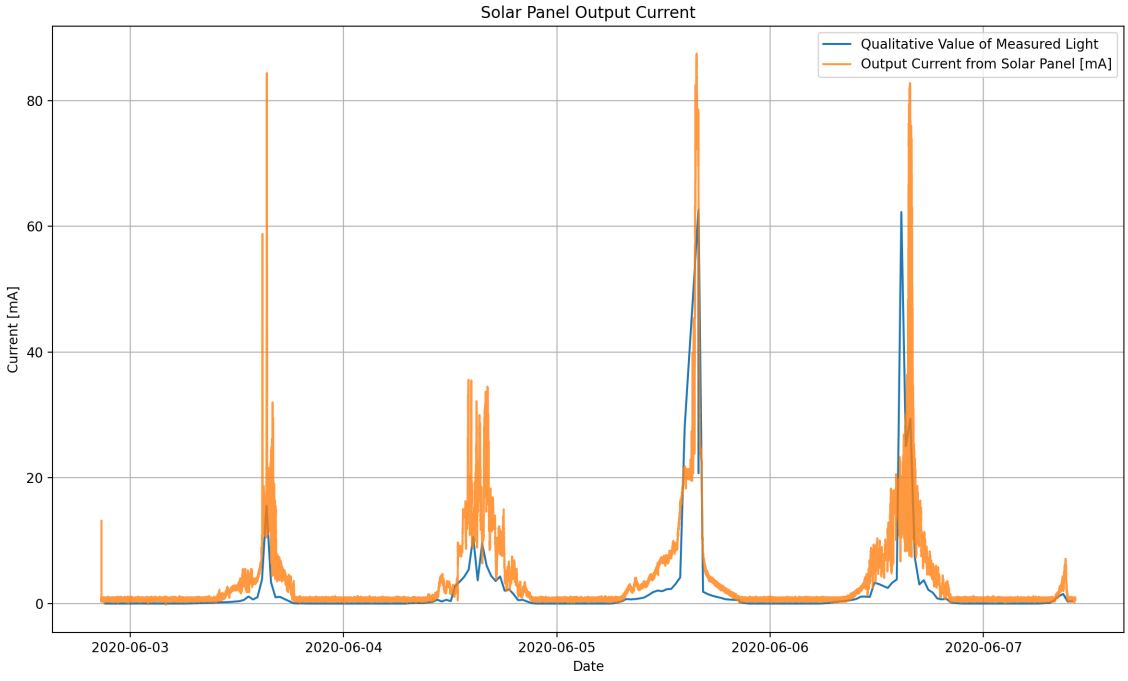


Figure 4.3: Solar panel output current.

It is important to note the fact that the solar panel is installed indoors, therefore remaining protected from the environmental elements. Additionally, the window has a metal fence in front of it that produces a vertically striped shadow, which also limits the amount of light the panel receives. The shadow effect on the panel is translated as a periodic oscillation in the output current, which is depicted in Figure 4.4.

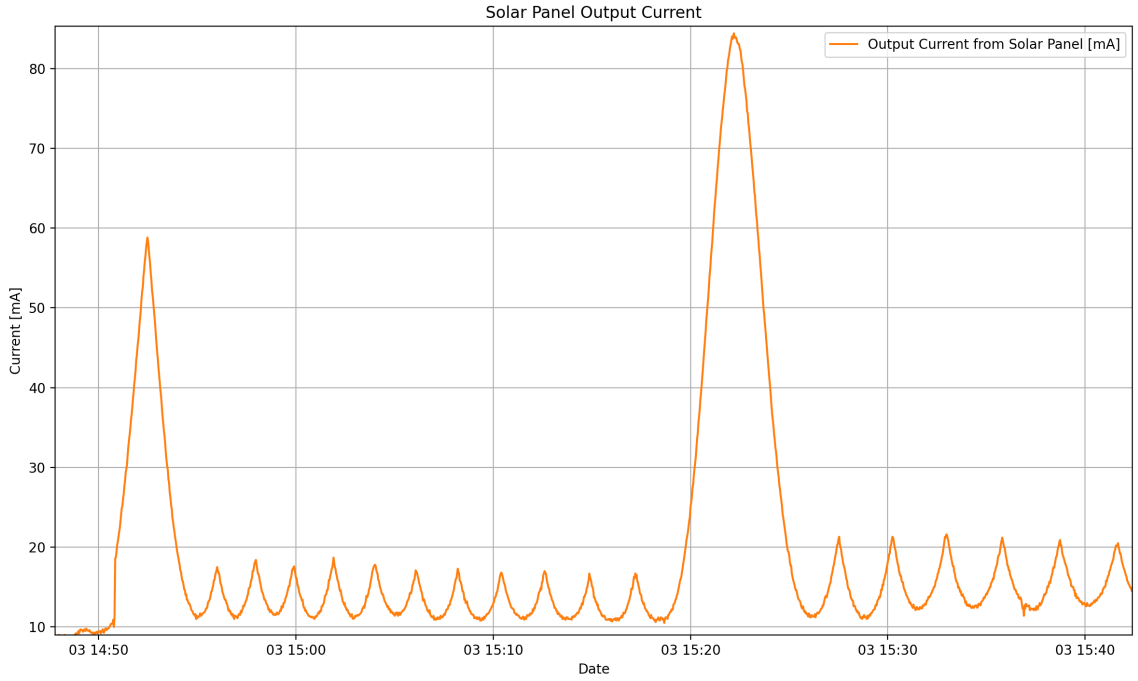


Figure 4.4: Zoomed in plot of the output current.

4.2. Software Description

In order to understand the obtained simulation results (Section 4.3) and the changes performed to the simulation (Section 4.4), a brief explanation on how the simulation is generated must be done.

Each timestep, the simulation uses the previous known states to find the following temperature value. In Equation 4.2, a simplified version of the recursive equation that is being calculated for each timestep k can be found.

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + B_{\mathrm{v}}\mathbf{v}(k) \tag{4.2}$$

where $\mathbf{x}(k)$ donotes the states, $\mathbf{v}(k)$ the external disturbances at the time instance t_k, and A and B_{v} are the correlation matrices. Two consecutive time instances t_k and

t_{k+1} differ by a timestep Δt such that $t_{k+1} = t_k + \Delta t$.

The external disturbances vector $\mathbf{v}(k)$ contains information about the ambient and ground temperatures, the global solar irradiance and the internal gains[1], and it is generated from the thermal model and the weather data.

The matrices A and B_v represent the time-invariant model parametrization between each of the elements of the building, which is obtained through the BRCM modelling.

4.3. Software Evaluation

This section compares the real measurements with the simulated data obtained with the *pybrcm* software to examine how accurate the simulation is. In this regard, it is important to specify the assumptions and context in which this comparison takes place.

First of all, the specific location of the devices has proven to be especially relevant, even though both devices are located in rooms within the same residence. The device 12 is located in a room with a window facing south-west, therefore having potential direct sunlight exposure. On the other hand, the device 3 is installed in a room with a small window that connects to a lightwell[2].

Secondly, the sensor data samples are taken in both devices every 30 minutes (Δt = 30 minutes) during a period of 62 days[3].

Additionally, since there is no available geometrical data for the building where the measures are taken, data from a building of Berlin with a similar distribution is imported.

In order to compare both simulations, two different approaches are pursued: a visual comparison and a statistical analysis. Therefore, a Python script is created that can plot the temperature measured for each of the two devices and the simulated one for the specified date range, and also performs a basic statistical analysis of the data.

Table 4.2 summarizes the average temperature and typical deviation from both devices during the same period of time. As it can be seen, the device 12, located in the room with a wall shared with the exterior, shows 0.71 degrees lower average temperature and notably higher fluctuations in its temperature value. On the other

[1] The *internal gains* model heat gains due to occupants, lighting and appliances.
[2] A *lightwell* is a space provided within the volume of a building for ventilation purposes.
[3] The measurements are made between May 7$^{\mathrm{th}}$ and June 9$^{\mathrm{th}}$.

hand, the device 3, located in the room without any walls shared with the exterior, shows a different behaviour. The same behaviour can be appreciated in a zoomed in plot of the temperature measurements in Figure 4.5.

	Device 03	Device 12
Avg. Temperature [°C]	22.5	21.79
Standard Deviation [°C]	1.53	1.94

Table 4.2: Average temperature and typical deviation from 7/4/20 until 14/5/20.

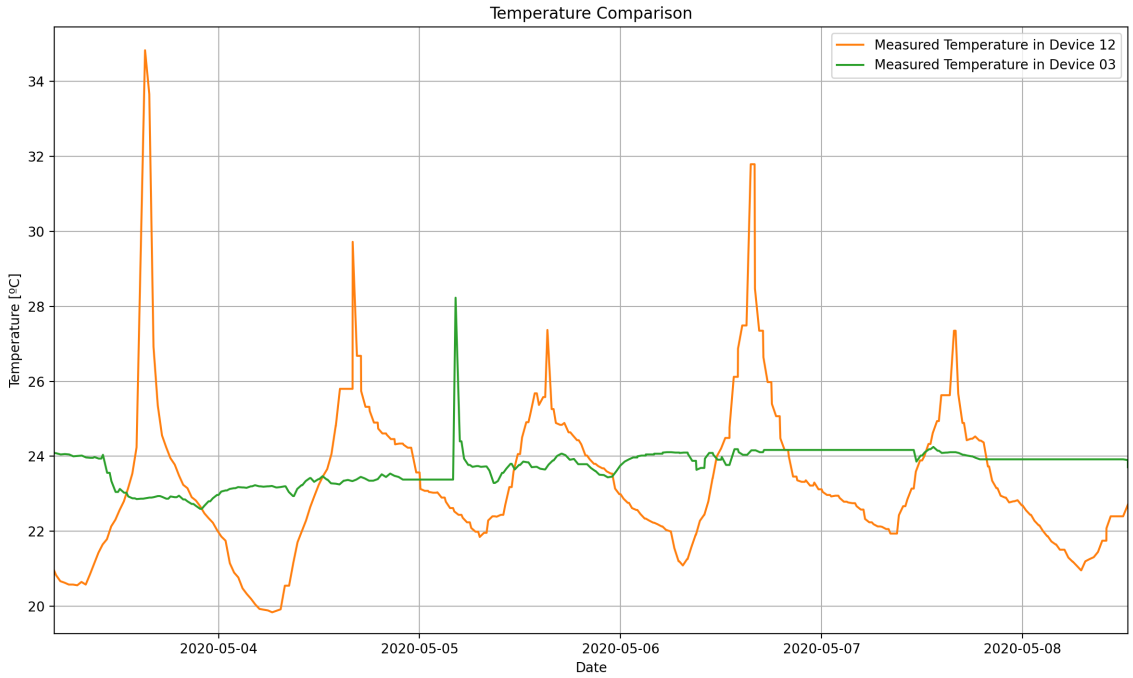


Figure 4.5: Zoomed in plot of the temperature measurements in device 3 and device 12.

In light of the above-mentioned results, before proceeding with the data evaluation, it is requisite to consider two different scenarios for each of the two rooms.

Scenario 1

This scenario models the room with a wall and a window shared with the exterior, where device 12 is located. Both the simulated and the measured temperatures are depicted in Figure 4.6.

The three days without measurements explained in Section 4.1.1, can also be appreciated in Figure 4.6 as a straight horizontal line. Even though the simulation starts

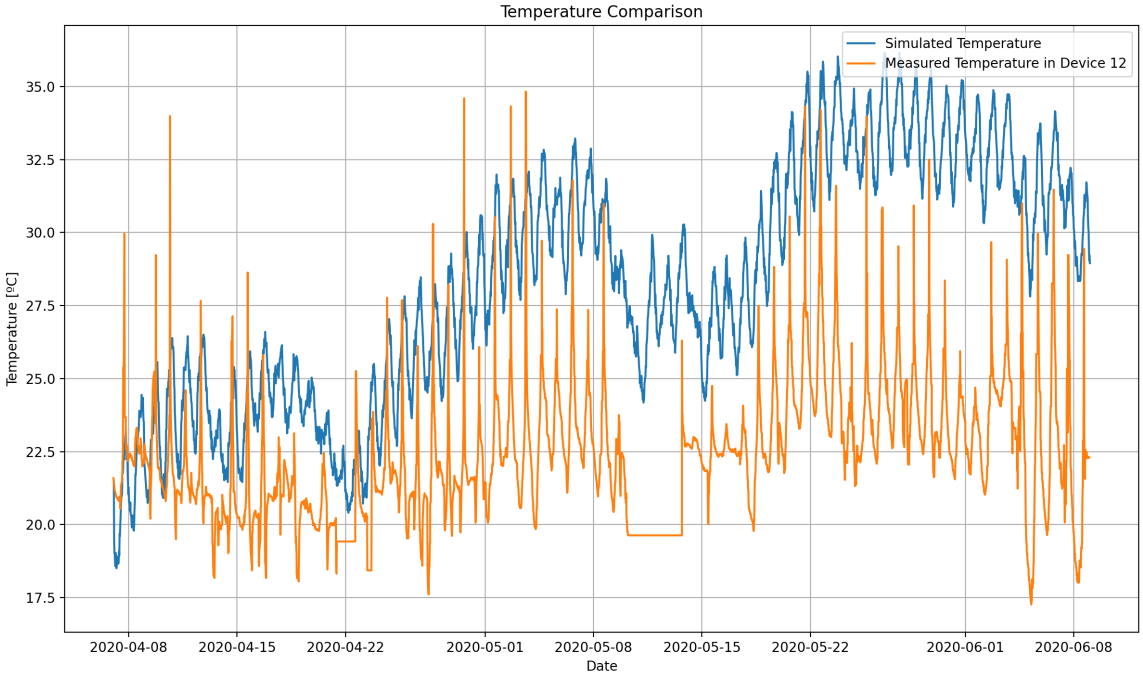


Figure 4.6: Measured temperature in device 12 (orange) and simulated temperature (blue).

with a notable resemblance with the measured indoor temperature, the simulated values keep increasing and rapidly become clearly above the comfort zone, even reaching temperatures of 36 ºC.

Unusually high temperature peaks are also measured almost each day. This behaviour, which is also seen by other Pysense users [11, 12], is probably caused by a design flaw of the Pysense PCB, which absorbs too much heat from the solar radiation and from the connected LoPy4 module.

Scenario 2

This scenario models the room without walls shared with the outside, having only one small window connecting to a lightwell. In Figure 4.7, a similar situation as in Scenario 1 can be observed, obviously taking into account the fact that the measured temperature in this room remains more constant.

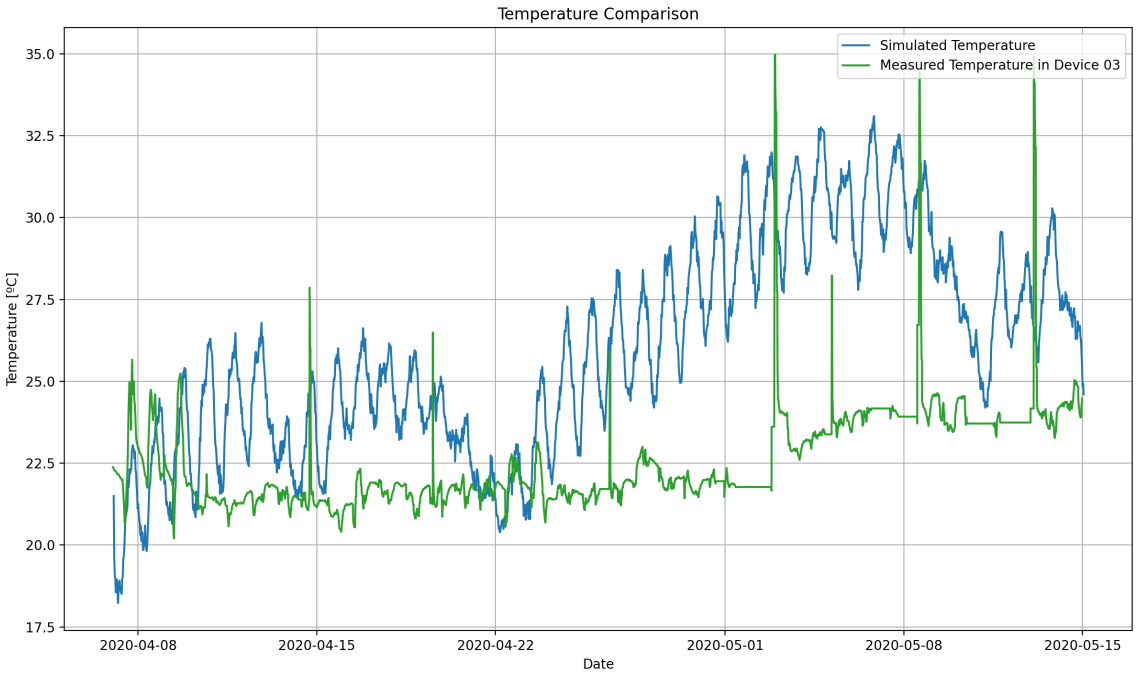


Figure 4.7: Measured temperature in device 3 (green) and simulated temperature (blue).

The data range for this scenario is notably reduced compared to the previous scenario[1] because from May 16th, the device 3 disconnects three times due to maintenance, battery exhaustion and malfunction, which crucially affects the continuity of the measurements. Therefore, all data from May 16th until June 9th is neglected .

Seven unusually high temperature peaks can also be appreciated. Since the sensor in this room has no direct sunlight exposure (therefore less heat absorption), it it reasonable to see this event less frequently compared to the Scenario 1.

[1]In Scenario 1 the data range goes from 07/04/20 until the 09/06/20 (64 days), in Scenario 2 goes from 07/04/20 until 15/05/20 (39 days).

4.4. Software Modification

Scenario 1

A high correlation can be appreciated between the simulated and ambient temperatures, whereas in the measured temperature this correlation is not as prevalent. To reduce it, the weight of the external disturbances is decreased. Through trial and error, a coefficient of 0.7942 is multiplied with the B_v product to obtain a simulation with the exact same average temperature as the measured one (See Equation 4.3). Figure 4.8 depicts the new simulation compared to the temperature measurements, which are now considerably more akin compared to Figure 4.6.

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + 0.7942 \cdot B_v\mathbf{v}(k) \tag{4.3}$$

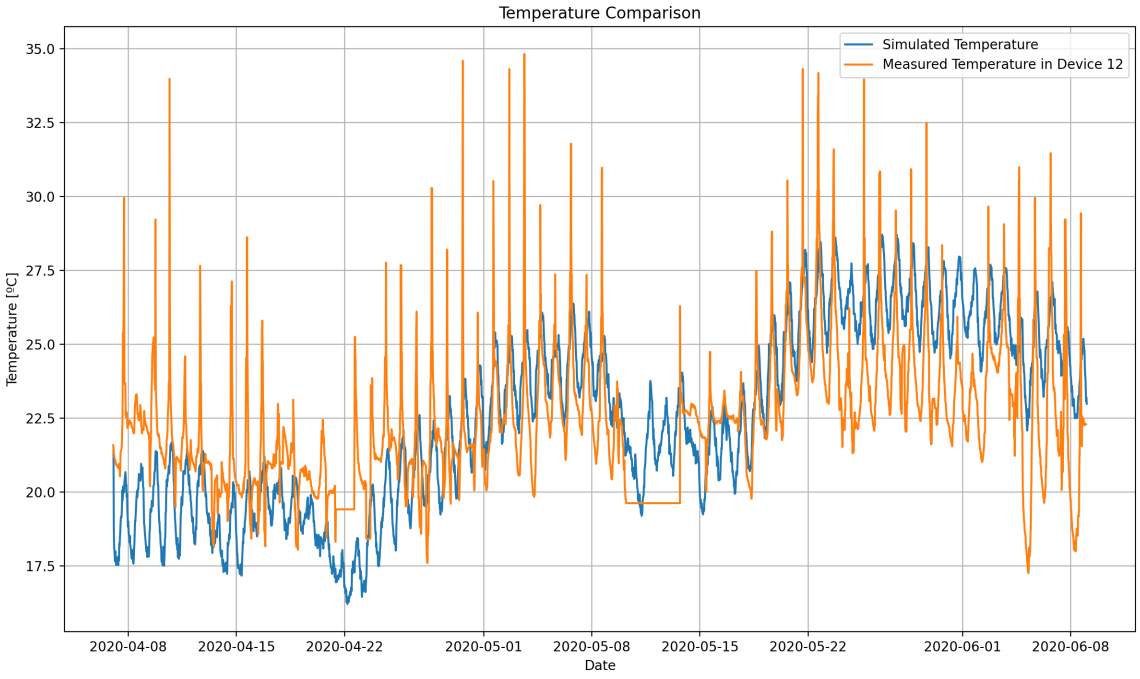


Figure 4.8: Measured temperature in device 12 and modified simulated temperature.

Scenario 2

In scenario 2, it is important to note that in the room where the device 3 is located, since there is almost no contact with the exterior, the indoor temperature is almost independent from the day and night cycle. For this reason, the dependence with the external disturbances needs to be further decreased until a 70% of the original value (See Equation 4.4).

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + 0.7 \cdot B_{\mathrm{v}}\mathbf{v}(k) \tag{4.4}$$

Additionally, the building predetermined rulesets in the *citygml2brcm* package are also modified to suit the particular characteristics for this scenario. Firstly, the average window to wall ratio and the window heat transfer coefficient are reduced by a 90% and a 30% respectively, to simulate the fact that this room has practically no window. Secondly, the walls heat capacity and density were increased by a 50 % to simulate the absence of walls shared with the exterior. Finally, the heat transfer coefficients[1] were also decreased by a 30% to retain a more constant temperature in the room. Figure 4.9 represents the comparison between the measured and modified simulated temperature.

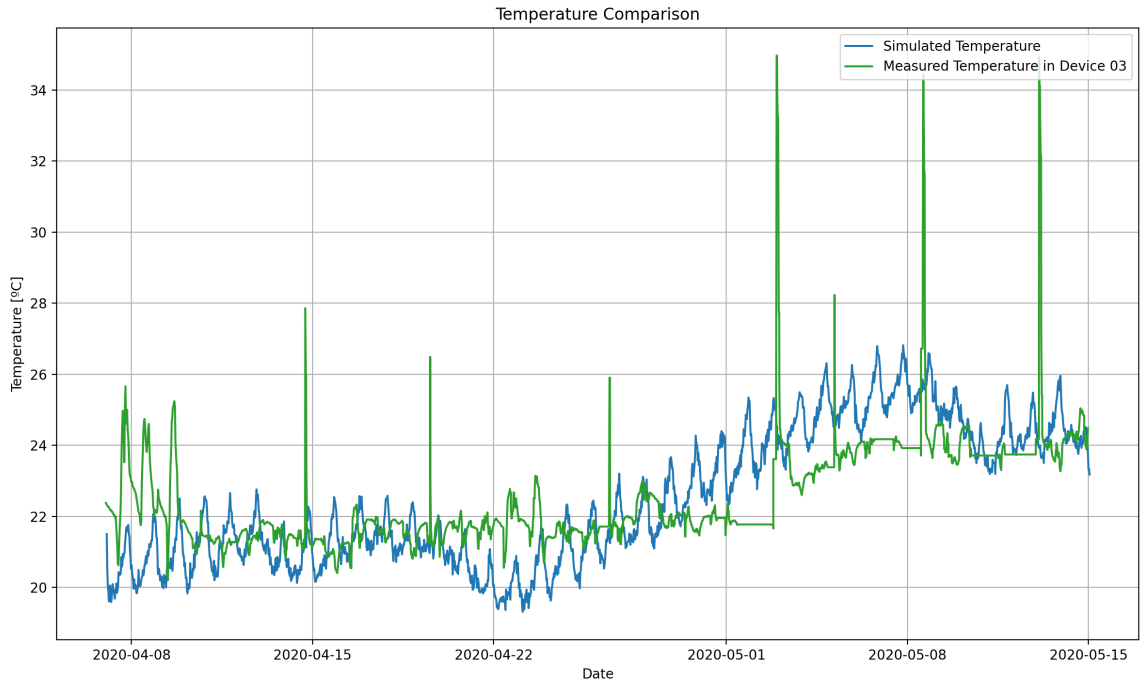


Figure 4.9: Measured temperature in device 3 and the modified simulated temperature.

[1]The heat transfer coefficients, often also referred as U-values, are used to measure how effective elements of a building are as insulators. The lower the coefficient, the more slowly heat is able to transmit through it, and so the better it performs as an insulator.

Even though the modified simulation fits the measured data considerably better than before, it still oscillates excessively. The reason why this oscillations cannot be further decreased is that the software's temperature simulation algorithms are highly tied to the day and night cycle, which is the case for most real scenarios, but not this one.

Chapter 5

Conclusions

5.1. Conclusions

Just like this thesis is founded on the work of other students, the work done also aims to become a useful tool for future development and investigation. For this reason, a lot of effort is focused on preparing the setup for future applications, e.g. the sketches for protective cases can be downloaded and printed at any point, the website is designed to work for any city, the battery life of the devices is optimized, etc.

There are crucial assumptions and simplifications which could be further investigated, such as the fact that the building's geometrical data imported for the simulations does not correspond to the building where the actual measurements are made.

Simulating a real scenario to its exact detail has always been an extremely difficult task, but the software created by the Laboratory of Internet of Things for Smart Buildings has proven to be exceptionally configurable and adjustable. The level of parameters which can be modified to approach the reality in each individual scenario is immense.

In this extent, the final results obtained from the modified simulation are satisfactory. On the other hand, there is improvement potential especially for the simulation of the scenario 2. Since the software always simulates minimal temperature oscillations for the indoor temperatures in compliance with the day and night cycle, the software's algorithms could be adjusted to replicate the measured almost constant temperature behaviour in its totality.

Moreover, the results are obtained in a situation where the number of people and time spent in the residence has been unusually high because of the mandatory lock-

down. This behaviour is probably reflected in higher overall temperatures, making the models not particularly representative of a *normal* situation.

5.2. Further Work

There are several ways along which the current project can be improved. Some of the future lines of development could be the following:

- ◇ Following the original plans for this project, install the sensors in the Robert Koch Forum in Berlin to study the temperature management in a public institution's building.
- ◇ Further extend the battery life by modifying the solar panel case to make it resistant to climatological adversities, so that it could be installed outdoors to maximize its sun exposure.
- ◇ Add more features to the created website, such as adding a functionality to also display the weather from other cities or to perform a basic statistical analysis with the measured data.

Chapter 6

Budget

This chapter exposes the costs involved in the development of the project. The two main contributors to the total cost are materials, maintenance and labor.

Materials

Table 6.1 discloses the price of the materials used for the development of the project.

Concept	Units	Unitary Price	Total Price
LoPy4	2	34.95 €	69.9 €
Pysense	2	24.95 €	49.9 €
Temperature Sensor (DS18B20)	2	1.9 €	3.8 €
Indoor Gateway	1	79.5 €	79.5 €
Battery (DTP634169)	2	2.85 €	5.7 €
Antenna	2	10 €	20 €
Solar Panel (CNC110X60)	1	4.1 €	4.1 €
Solar Manager (DFR0264)	1	4.5 €	4.5 €
DC Current Sensor (INA219)	1	6.99 €	6.99 €
TOTAL			**244.39 €**

Table 6.1: Material list.

Maintenance

The most notable maintenance price is the Amazon Web Services fee for the webpage hosting. The monthly price of the service is 5 €.

Labor

A wage for a undergraduate student has been taken into consideration. The wage breakdown is detailed in Table 6.2, and it already includes social security taxes.

	Dedication	Wage/hour	Total hours	Total wage
Undergraduate engineer	27 h/week	9 €	450 hours	4050 €

Table 6.2: Labor Costs.

Bibliography

[1] European Comission. Energy performance of buildings directive [Online]. https://ec.europa.eu/energy/topics/energy-efficiency/energy-efficient-buildings/energy-performance-buildings-directive_en. Accessed: 14/06/2020.

[2] ASHRAE Standards Committee 2009-2010. Thermal environmental conditions for human occupancy (standard 55-2010). Technical report, 2010.

[3] Marcos Torres Padín. A lora-based framework with applications to smart buildings. Master's thesis, Universitat Politècnica de Catalunya, March 2020.

[4] Danny Nowka. Automatic thermal modeling of city districts based on open data for energy-optimal predictive control. Master's thesis, Technische Universität Berlin, July 2019.

[5] Alexander Batoulis. Optimal operation of a smart building based on intelligent algorithms and a digital twin. Master's thesis, Technische Universität Berlin, October 2018.

[6] Swagger. The Things Network Data Storage [Online]. https://smart_buildings.data.thethingsnetwork.org/. Accessed: 12/06/2020.

[7] Agencia Estatal de Meteorología. AEMET Open Data (Spanish) [Online]. https://opendata.aemet.es/centrodedescargas/productosAEMET? Accessed: 10/06/2020.

[8] Deutscher Wetterdienst. Climate Data Center [Online]. https://opendata.dwd.de/climate_environment/CDC/. Accessed: 10/06/2020.

[9] Adel Ghazel Martin Murnane. A closer look at state of charge (soc) and state of health (soh) estimation techniques for batteries. *Analog Devices*, page 8, 2017.

[10] Caiping Zhang, Jiuchun Jiang, Linjing Zhang, Sijia Liu, Leyi Wang, and Poh Chiang Loh. A generalized soc-ocv model for lithium-ion batteries and the soc estimation for lnmco battery. *Energies*, 9(11):900, 2016.

[11] GaneshVarahade. Error Temperature Reading [Online]. https://github.com/pycom/pycom-micropython-sigfox/issues/156. Accessed: 25/06/2020.

[12] bmarkus. PySense accuracy [Online]. https://forum.pycom.io/topic/2001/pysense-accuracy. Accessed: 25/06/2020.

Appendix A

Casing and Setup Pictures

A.1. Sensor Casing

This section contains pictures of the 3D-cases printed for each component of the sensor setup. The case for the LoPy4 and the Pysense boards is depicted in Figure A.1. The case for the external temperature sensor is shown in Figure A.2, which is connected to the Pysense board through three braided cables (Figure A.3).

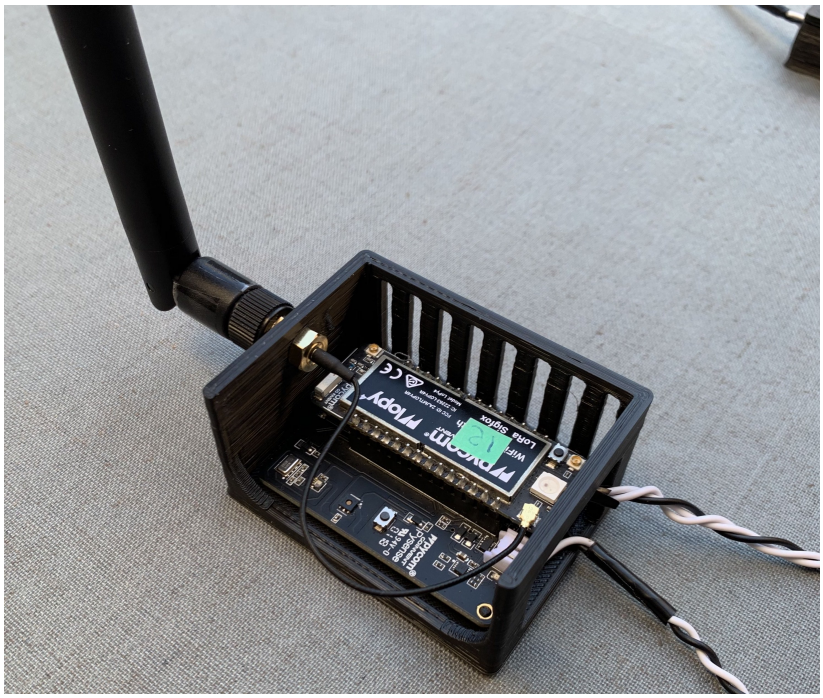
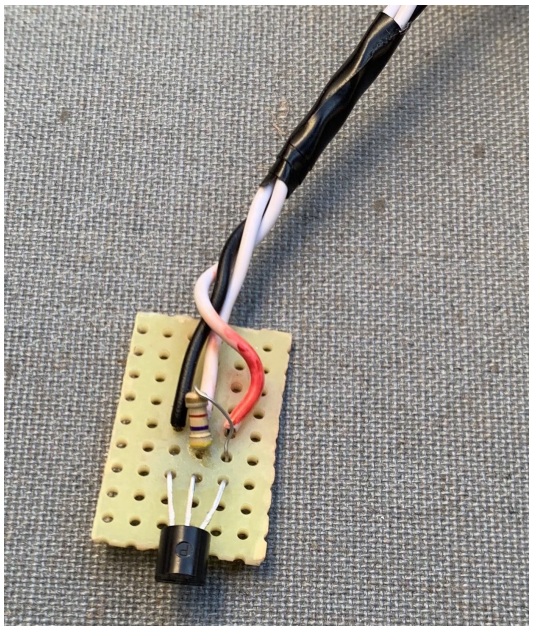


Figure A.1: Printed case for the Pysense and LoPy4.



(a) External temperature sensor.

(b) Casing for the temperature sensor.

Figure A.2: External temperature sensor setup.

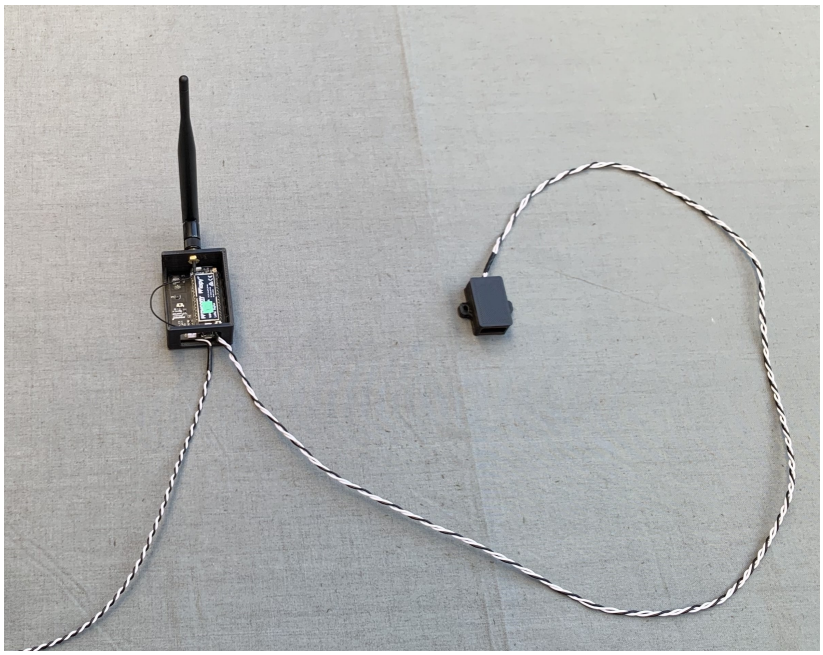


Figure A.3: Encased sensor connected to the external temperature sensor.

The external temperature sensor installed in the heater to measure its activity is shown in Figure A.4, and the whole setup including the solar panel is depicted in Figure A.5.

Figure A.4: External temperature sensor attached to the heater.

Figure A.5: Whole setup including solar panel.

A.2. Webpage Screenshots

Figure A.6 is a screenshot of the developed webpage. The measured parameters can be selected to be shown or not for the specified device, as well as the weather forecast. The date range to be displayed can also be selected through a calendar window, which is shown in Figure A.7. The webpage can be accessed through the following URL.

`http://ec2-34-228-44-224.compute-1.amazonaws.com/`

Figure A.6: Webpage main interface screenshot.

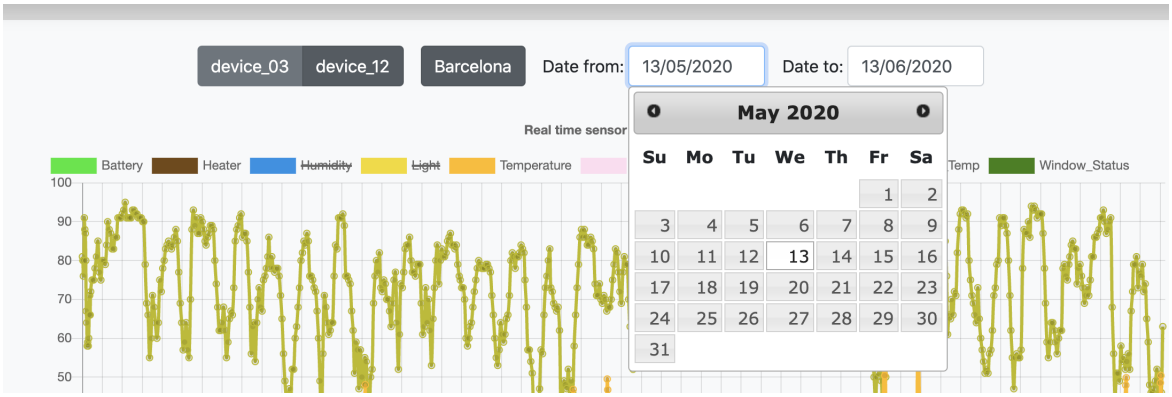


Figure A.7: Date selector window.

A.3. Solar Panel Structure

The structure for the solar panel (Figure A.8) has a movable joint, so that it can be oriented to maximize sun exposure. The solar charger can be attached to the structure (Figure A.9), and it also has a small compartment to protect the battery (Figure A.10). The ouput of the solar charger is directly connected to the power supply of the sensor board (Figure A.11).

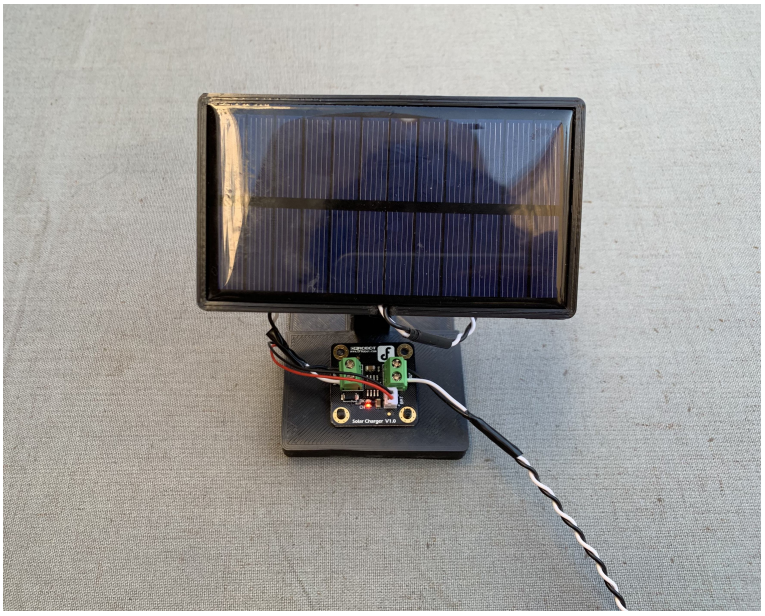


Figure A.8: Adjustable support for the solar panel: front.

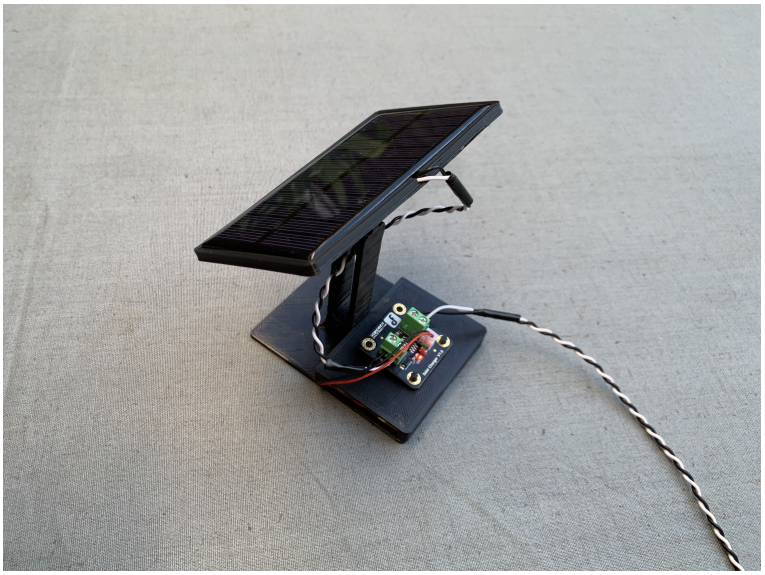


Figure A.9: Adjustable support for the solar panel: back.

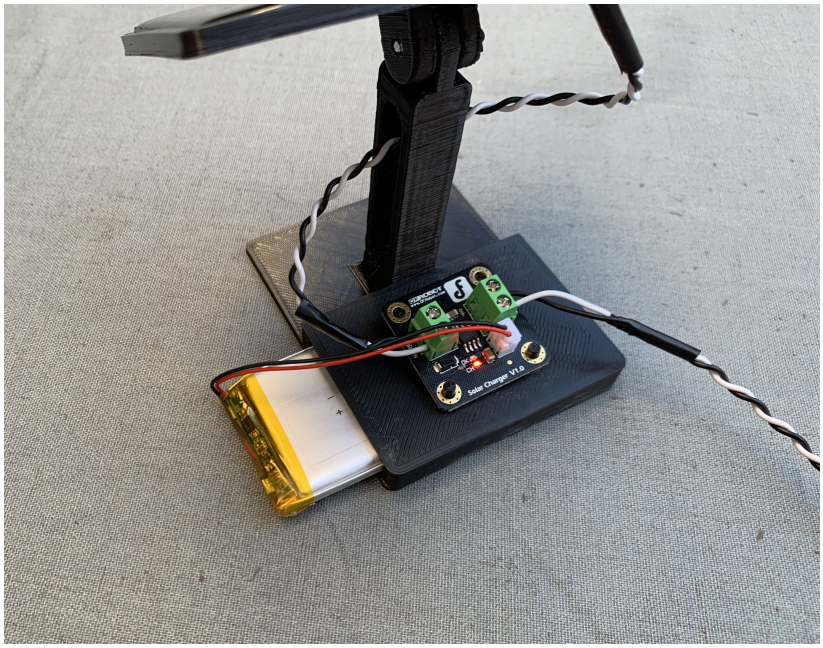


Figure A.10: Battery casing.

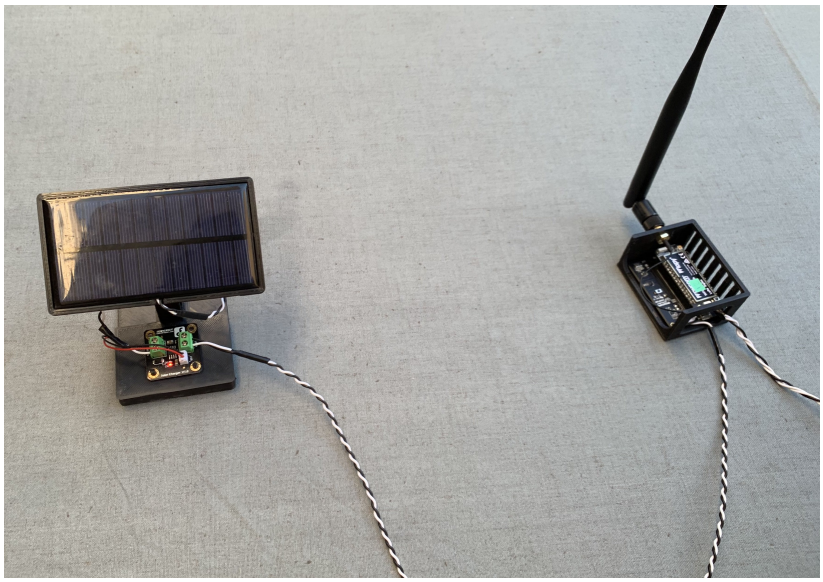


Figure A.11: Solar panel connected to the sensor board.

Appendix B

Python Code

B.1. *plot_comparison* Script

This script runs the simulation for the specified data range, and then plots the output along with the measured data from the selected devices to visually compare the results. It also performs a basic statistical analysis.

```
import sys
import pickle
import numpy as np
import datetime as dt
import logging
import json
import csv
logging.basicConfig(level = logging.INFO)

# include citygml2brcm and pybrcm
sys.path.append("./../02_citygml2brcm")
from pybrcm.Simulation import Simulation
import pybrcm.Evaluation as Evaluation

# define the input files
buildingsFile   = "../90_output/tfg/buildings.pickle"
modelFile       = "../90_output/tfg/model_Ts30.mat"
weatherFile     = "../90_output/weather_conv/weather_20200402-210000
    _to_20200609-130000.csv"
sensorDataFile_03  = "../30_sensor_data_scripts/device_03_all_data.json"
sensorDataFile_12  = "../30_sensor_data_scripts/device_12_all_data.json"
sensor_data_base_dir = "../30_sensor_data_scripts/device_{}_all_data.json"


# create the simulation environment
sim = Simulation(modelFile, weatherFile, buildingsFile)
```

```
26
27 # define simulation time range
28 dtbeg = dt.datetime(2020, 4, 7, 0, 0, 0)
29 dtend = dt.datetime(2020, 6, 8, 23, 59, 59)
30 device_list = ('device_03', 'device_12')
31
32 # run simulation
33 res = sim.run(x0 = 21.5, dtBegin = dtbeg, dtEnd = dtend)
34
35 import matplotlib.pyplot as plt
36
37 def read_data(device_id):
38     directory = sensor_data_base_dir.format(device_id)
39     print('Reading:', directory)
40     with open(directory) as f:
41         data = json.load(f)
42     return data
43
44
45 def read_csv_file(filename, cols):
46     time_list = list()
47     current_list = list()
48     with open(filename, 'r') as f:
49         line_reader = csv.reader(f, delimiter=';')
50         for row in line_reader:
51             time_list.append(int(row[0]))
52             current_list.append(float(row[1]))
53     return time_list, current_list
54
55
56 def change_date_format(data):
57     for i in range(len(data)):
58         data[i]['time'] = dt.datetime.strptime(data[i]['time'].split('Z')[0],
     "%Y-%m-%dT%H:%M:%S")
59     return data
60
61
62 def get_device_id_from_device_name(device_name):
63     return device_name.split('_')[1]
64
65
66 def filter_data_through_dates(data, dtbeg, dtend):
67     return list(filter(lambda x: x['time'] >= dtbeg and x['time'] <= dtend,
    data))
68
69
70 def filter_data_range(data, dtbeg, dtend):
71     first_date = data[0]['time']
72     last_date = data[-1]['time']
73     if dtbeg<first_date or dtend>last_date:
```

```
74          raise Exception("Imported sensor data dates are not within selected
    range")
75      else:
76          filtered_data = filter_data_through_dates(data, dtbeg, dtend)
77
78      return filtered_data
79
80
81  def subplot_current():
82      directory = '../30_sensor_data_scripts/current_measurements'
83      filename = 'solar_panel_current.csv'
84      file_dir = directory + '/' + filename
85      cols = ['secs', 'current_mA']
86
87      time_list, current_list = read_csv_file(file_dir, cols)
88      plt.plot(time_list, current_list)
89
90
91  def plot_all(data):
92      plt.title('Temperature Comparison')
93      subplot_temp(data)
94      plt.grid()
95      plt.xlabel('Date')
96      plt.ylabel('Temperature [ C]')
97      plt.show()
98
99
100 def subplot_light(data):
101
102     for device_name in device_list:
103         device_id = get_device_id_from_device_name(device_name)
104         dates = [d['time'] for d in data[device_id]]
105         unnormalized_light = [d['light'] for d in data[device_id]]
106         plt.plot(dates, unnormalized_light,
107             label = f'Measured Sensor Light in Device {device_id}')
108
109     plt.legend(loc = 'upper left', fontsize = 'x-small')
110
111
112 def plot_weekday_line(data, device_id):
113     weekdays = []
114     for d in data[device_id]:
115         wd =  d['time'].weekday()
116         if wd in (0, 1, 2, 3, 4):
117             weekdays.append(int('10'))
118         else:
119             weekdays.append(int('20'))
120
121     dates = [d['time'] for d in data[device_id]]
122     plt.plot(dates, weekdays, label='Weekend days',color='purple', linewidth
    =2, alpha=0.5)
```

```
123
124
125 def subplot_temp(data):
126     prop_cycle = plt.rcParams['axes.prop_cycle']
127     colors = prop_cycle.by_key()['color']
128
129     zone_temp_values = np.mean(res["xZone"], axis=0)
130     plt.plot(res["tloc"], zone_temp_values, label = "Simulated Temperature",
        color=colors[0])
131
132
133     for device_name in device_list:
134         device_id = get_device_id_from_device_name(device_name)
135         dates = [d['time'] for d in data[device_id]]
136         y = [d['temperature'] for d in data[device_id]]
137         plt.plot(dates, y, label=f'Measured Temperature in Device {device_id}
        ')
138
139     plt.legend(loc = 'upper right', fontsize = 'medium')
140
141
142 def temp_in_weekday_and_weekend(data, device_id):
143     temp_wd = []
144     temp_we = []
145
146     for d in data[device_id]:
147         wd =  d['time'].weekday()
148         if wd in (0, 1, 2, 3, 4):
149             temp_wd.append(d['temperature'])
150         else:
151             temp_we.append(d['temperature'])
152
153     return (np.average(temp_wd), np.average(temp_we))
154
155
156 def print_temperature_values(data):
157     print('

    ')
158     print('                          MEAN TEMPERATURES:                          '
    )
159     print('

    ')
160
161     avg_amb = float(np.mean(res['Tamb']))
162     print('Average Ambient Temperature:                          |   '+'%.2f' %
    avg_amb + '    |')
163
164     zone_temp_values = np.mean(res["xZone"], axis=0)
165     avg_zone = np.mean(zone_temp_values)
```

```
166     print('Average Zone Temperature:                                    |  '+'%.2f' %
    avg_zone + '    |')
167
168     for device_name in device_list:
169         temperatures = []
170         device_id = get_device_id_from_device_name(device_name)
171         for d in data[device_id]:
172             temperatures.append(d['temperature'])
173         avg_measured = np.mean(temperatures)
174         temp_weekdays = temp_in_weekday_and_weekend(data, device_id)
175         print('

    |                                |')
176         print(f'Average Measured Temp. in Device {device_id}:
     |  '+'%.2f' %avg_measured + '    |')
177         print(f'Average Weekday Temp. in Device {device_id}
     |  '+'%.2f' %temp_weekdays[0] + '    |')
178         print(f'Average Weekend Temp. in Device {device_id}:
     |  '+'%.2f' %temp_weekdays[1] + '    |')
179
180
181 def filter_unwanted_peaks(data):
182     threshold_temperature = 40
183     for i in range(len(data)):
184         if (data[i]['temperature'] == None):
185             data[i]['temperature'] = data[i-1]['temperature']
186         elif (data[i]['temperature'] > threshold_temperature):
187             data[i]['temperature'] = data[i-1]['temperature']
188
189     return data
190
191
192 def statistical_analysis(data):
193     import statistics
194     print('

    ')
195     print('                        STATISTICAL ANALYSIS:                       '
    )
196     print('

    ')
197
198     zone_temp_values = np.mean(res["xZone"], axis=0)
199     stdev = statistics.stdev(zone_temp_values)
200     variance = statistics.variance(zone_temp_values)
201     print(f'Standard deviation of Zone temperature:              |  ' + '%.2f
    ' %stdev + '    |')
202     print(f'Variance of Zone temperature :                       |  ' + '%.2f
    ' %variance + '    |')
203
```

```
    for device_name in device_list:
        temperatures = []
        device_id = get_device_id_from_device_name(device_name)
        for d in data[device_id]:
            temperatures.append(d['temperature'])

        stdev = statistics.stdev(temperatures)
        variance = statistics.variance(temperatures)

        print('
    |                                |')
        print(f'Standard deviation of temperature in {device_name}       |  '
    + '%.2f' %stdev + '     |')
        print(f'Variance of temperature in {device_name}:                |  '
    + '%.2f' %variance + '    |')


def evaluate_simulation(data):
    print_temperature_values(data)
    statistical_analysis(data)

def main():
    data = dict()
    for device_name in device_list:
        device_id = get_device_id_from_device_name(device_name)
        data_read = read_data(device_id)
        formatted_data = change_date_format(data_read)
        data[device_id] = filter_data_range(formatted_data, dtbeg, dtend)
        data[device_id] = filter_unwanted_peaks(data[device_id])
    plot_all(data)
    evaluate_simulation(data)


if __name__ == "__main__":
    main()
```

Listing B.1: Plot Comparison Script

B.2. *plot_battery_evolution* Script

This script is used to evaluate the battery performance. It plots the battery voltage along with the fitted lines for each of the discharging periods.

```
import datetime as dt
import json
import numpy as np
import matplotlib.pyplot as plt

sensor_data_base_dir = "device_{}_all_data.json"
device_list = ['device_12']

def set_dates(data):
    dtbeg = dt.datetime(2020, 4, 3, 14, 0, 0)
    last_days = []
    for device_name in device_list:
        device_id = get_device_id_from_device_name(device_name)
        last_days.append(data[device_id][-1]['time'])

    return dtbeg, min(last_days)


def read_data(device_id):
    directory = sensor_data_base_dir.format(device_id)
    print('Reading:', directory)
    with open(directory) as f:
        data = json.load(f)
    return data


def change_date_format(data):
    for i in range(len(data)):
        data[i]['time'] = dt.datetime.strptime(
            data[i]['time'].split('Z')[0], "%Y-%m-%dT%H:%M:%S")
    return data


def filter_data_through_dates(data, dtbeg, dtend):
    return list(filter(lambda x: x['time'] >= dtbeg and x['time'] <= dtend,
    data))


def filter_data_range(data, dtbeg, dtend):
    first_date = data[0]['time']
    last_date = data[-1]['time']
    if dtbeg < first_date or dtend > last_date:
        raise Exception(
            "Imported sensor data dates are not within selected range")
    else:
```

```
45          filtered_data = filter_data_through_dates(data, dtbeg, dtend)
46
47      return filtered_data
48
49
50  def get_device_id_from_device_name(device_name):
51      return device_name.split('_')[1]
52
53
54  def filter_data_type(data, fields):
55      for i, d in enumerate(data):
56          data[i] = {field: d[field] for field in fields}
57
58      return data
59
60
61  def filter_unwanted_battery_values(data, method, d_periods):
62      for discharge_key in d_periods:
63          threshold_date_lb =  d_periods[discharge_key][0]
64          threshold_date_ub =  d_periods[discharge_key][1]
65          threshold_voltage = 4.25
66
67          if method == 'relative':
68              for i in range(len(data)):
69                  if data[i]['time'] > threshold_date_lb:
70                      avg = 0
71                      if i < (len(data)-5):
72                          for j in range(5):
73                              avg += data[i+j]['battery']
74                          avg = avg/5
75                      if i > 0 and (data[i]['battery'] > avg):
76                          data[i]['battery'] = data[i-1]['battery']
77
78          elif method == 'absolute':
79              for i in range(len(data)):
80                  if (data[i]['battery'] == None):
81                      data[i]['battery'] = data[i-1]['battery']
82                  if (data[i]['time'] > threshold_date_lb and
83                      data[i]['time'] < threshold_date_ub and
84                      float(data[i]['battery']) > threshold_voltage):
85
86                      data[i]['battery'] = data[i-1]['battery']
87
88          else:
89              raise Exception(
90                  "Selected battery filtration method is not valid")
91      return data
92
93
94  def define_discharging_periods(dtbeg, dtend):
95      periods_03 = {
```

```
        'discharge_1': [dtbeg,
                        dt.datetime(2020, 4, 14, 14, 0, 0)],

        'discharge_2': [dt.datetime(2020, 4, 15, 0, 0, 0),
                        dt.datetime(2020, 5, 1, 10, 0, 0)],

        'discharge_3': [dt.datetime(2020, 5, 3, 12, 0, 0),
                        dtend]
    }
    periods_12 = {
        'discharge_1': [dtbeg,
                        dt.datetime(2020, 4, 14, 14, 0, 0)],

        'discharge_2': [dt.datetime(2020, 4, 15, 0, 0, 0),
                        dt.datetime(2020, 5, 10, 2, 0, 0)],

        'discharge_3': [dt.datetime(2020, 5, 13, 18, 0, 0),
                        dtend]
    }

    periods = {
        '03': periods_03,
        '12': periods_12
    }

    return periods


def get_slope(data, dtbeg, dtend):
    battery_level = []
    dates = []
    for i,d in enumerate(data):
        if d['time'] >= dtbeg and d['time'] < dtend:
            battery_level.append(float(d['battery']))
            dates.append(i)
    slope = np.polyfit(dates, battery_level, 1)
    slope = slope[0]

    return slope


def get_curve(data, slope, dtbeg, dtend):
    battery_level = []
    dates = []

    for d in data:
        if d['time'] >= dtbeg and d['time'] < dtend:
            battery_level.append(d['battery'])
            dates.append(d['time'])

    curve = [battery_level[0], slope, dates]
```

```
147
148     return curve
149
150
151 def analyse_battery(data, d_periods):
152     slopes = dict()
153     curves = dict()
154
155     for device_name in device_list:
156         device_id = get_device_id_from_device_name(device_name)
157         slopes[device_id] = {}
158         curves[device_id] = {}
159         for discharge in d_periods[device_id]:
160             slopes[device_id][discharge] = get_slope(
161                 data[device_id], d_periods[device_id][discharge][0],
    d_periods[device_id][discharge][1])
162             curves[device_id][discharge] = get_curve(
163                 data[device_id], slopes[device_id][discharge], d_periods[
    device_id][discharge][0], d_periods[device_id][discharge][1])
164
165     return curves
166
167
168 def plot_fitted_curves(data, curves, device_id):
169     for discharge_key in curves:
170         intercept = curves[discharge_key][0]
171         slope = curves[discharge_key][1]
172         dates = curves[discharge_key][2]
173         curve_to_plot = []
174         secs_elapsed = (dates[-1] - dates[0]).total_seconds()
175         days_elapsed = secs_elapsed / (24*3600)
176         print(days_elapsed)
177         for i in range(len(dates)):
178             curve_to_plot.append(intercept + slope*i)
179         slope_per_day = (curve_to_plot[-1] - curve_to_plot[0])/days_elapsed
180         slope_x1000 = slope_per_day * 1000
181         label = f'Slope of the fitted line: ' + '{:.2f}'.format(slope_x1000)
    + ' [mV/day]'
182         plt.plot([dates[0], dates[-1]], [curve_to_plot[0], curve_to_plot
    [-1]], alpha=0.7, label=label)
183
184
185 def plot_all(data, curves):
186     for device_name in device_list:
187         device_id = get_device_id_from_device_name(device_name)
188         battery_values = [d['battery'] for d in data[device_id]]
189         dates = [d['time'] for d in data[device_id]]
190         plt.plot(dates, battery_values,
191                  label=f'{device_name} battery voltage', linewidth=2)
192     for device_name in device_list:
193         device_id = get_device_id_from_device_name(device_name)
```

```
194         plot_fitted_curves(data[device_id], curves[device_id], device_id)
195
196     plt.grid()
197     plt.xlabel('Date')
198     plt.ylabel('Battery Voltage [V]')
199     plt.title('Battery Voltage Evolution')
200     plt.legend(loc='upper right')
201     plt.show()
202
203
204 def main():
205     data = dict()
206     fields = ('time', 'battery')
207     for device_name in device_list:
208         device_id = get_device_id_from_device_name(device_name)
209         data_read = read_data(device_id)
210         data[device_id] = change_date_format(data_read)
211
212     (dtbeg, dtend) = set_dates(data)
213     discharging_periods = define_discharging_periods(dtbeg, dtend)
214     for device_name in device_list:
215         device_id = get_device_id_from_device_name(device_name)
216         preprocessed_data = filter_data_range(data[device_id], dtbeg, dtend)
217         data[device_id] = filter_data_type(preprocessed_data, fields)
218         data[device_id] = filter_unwanted_battery_values(
219             data[device_id], method='absolute', d_periods=discharging_periods
    [device_id])
220     curves = analyse_battery(data, discharging_periods)
221     plot_all(data, curves)
222
223
224 if __name__ == '__main__':
225     main()
```

Listing B.2: Plot Battery Evolution Script

B.3. *automatic_data_retrieval* Script

This script accesses the specified API, downloads its content as a `*.json` file and updates a document with all the collected data.

```
import requests
from datetime import datetime, timedelta
import json
from time import sleep


def read_file(device_id):
    filename = f'{device_id}_all_data.json'
    with open(filename, 'r') as f:
        data = json.load(f)
    return data


def convert_to_datetime_from_api(date_string):
    return datetime.strptime(date_string.split('.')[0], "%Y-%m-%dT%H:%M:%S")


def convert_to_datetime_from_json(date_string):
    return datetime.strptime(date_string, "%Y-%m-%dT%H:%M:%S")


def change_timezone(date):
    date = date + timedelta(hours=2)
    return date


def convert_datetime_to_string(date):
    return date.strftime("%Y-%m-%dT%H:%M:%S")


def get_new_entries(old_data, new_data):
    to_append = []
    last_entry_date = convert_to_datetime_from_json(old_data[-1]['time'])
    for new in new_data:
        date = convert_to_datetime_from_api(new['time'])
        date = change_timezone(date)
        new['time'] = convert_datetime_to_string(date)
        if date > last_entry_date:
            to_append.append(new)
    print(f'Found {len(to_append)} new entries')
    return to_append


def append_new_data_to_file(device_id, old_entries, new_entries):
    all_data = old_entries + new_entries
```

```
    save_to_file(device_id, all_data)


def save_to_file(device_id, data):
    filename = f'{device_id}_all_data.json'
    with open(filename, 'w') as f:
        json.dump(data, f, indent=4)
    print(f'Saved to {filename}')


def query_last_day(device_id):
    headers = {
        'Accept': 'application/json',
        'Authorization': 'key ttn-account-v2.YV7-I4dNko-LKbawSsU0CdAZ0bqWpmtvgSjSPU0oYj4',
    }

    params = (
        ('last', '3d'),
    )

    url = f'https://smart_buildings.data.thethingsnetwork.org/api/v2/query/{device_id}'
    data = requests.get(url, headers=headers, params=params).json()
    return data


def main():
    ONE_HOUR = 60 * 60
    while True:
        devices = ['device_03', 'device_12']
        for device_id in devices:
            time_now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            print(f'{time_now}: Gathering info from {device_id}')
            old_entries = read_file(device_id)
            last_day = query_last_day(device_id)
            new_entries = get_new_entries(old_entries, last_day)
            append_new_data_to_file(device_id, old_entries, new_entries)
            print('Done\n')
        sleep(ONE_HOUR)


if __name__ == '__main__':
    main()
```

Listing B.3: Automatic Data Retrieval Script

B.4. *plot_current* Script

This script is used to visualize the current measurements obtained from the current sensor.

```
from time import sleep
import csv
import json
from datetime import datetime
import datetime as dt
import matplotlib.pyplot as plt

def read_csv_file(filename, cols):
    time_list = list()
    current_list = list()
    with open(filename, 'r') as f:
        line_reader = csv.reader(f, delimiter=';')
        for row in line_reader:
            time_list.append(int(row[0]))
            current_list.append(float(row[1]))
    return time_list, current_list


def read_data():
    directory = "device_12_all_data.json"
    print('Reading:', directory)
    with open(directory) as f:
        data = json.load(f)
    return data


def plot_current_data(time_list, current_list):
    prop_cycle = plt.rcParams['axes.prop_cycle']
    colors = prop_cycle.by_key()['color']

    plt.plot(time_list, current_list, label='Output Current from Solar Panel [mA]', color=colors[1])


def change_date_format(data):
    for i in range(len(data)):
        data[i]['time'] = dt.datetime.strptime(data[i]['time'].split('Z')[0], "%Y-%m-%dT%H:%M:%S")
    return data


def plot_light_data(data, dtbeg, dtend):
    dates_to_plot = list()
    light_to_plot = list()
    lights = list()
```

```
44
45      for d in data:
46          if (d['time'] > dtbeg and d['time'] < dtend and d['light'] is not
        None):
47              dates_to_plot.append(d['time'])
48              light_to_plot.append(d['light']/1000)
49
50      plt.plot(dates_to_plot, light_to_plot, label='Qualitative Value of
        Measured Light')
51
52
53  def convert_to_datetime(time_list):
54      initial_timestamp = 1591123440
55      for i in range(len(time_list)):
56          time_list[i] = datetime.fromtimestamp(initial_timestamp + time_list[i
        ])
57
58      return time_list
59
60
61  def main():
62      directory = 'current_measurements'
63      filename = 'solar_panel_current.csv'
64      file_dir = directory + '/' + filename
65      cols = ['secs', 'current_mA']
66
67      time_list, current_list = read_csv_file(file_dir, cols)
68      time_list = convert_to_datetime(time_list)
69      data = read_data()
70      data = change_date_format(data)
71      plot_current_data(time_list, current_list)
72
73      plt.grid()
74      plt.xlabel('Date')
75      plt.ylabel('Current [mA]')
76      plt.title('Solar Panel Output Current')
77      plt.legend(loc='upper right')
78      plt.show()
79
80
81  if __name__ == '__main__':
82      main()
```

Listing B.4: Plot Current Script

B.5. *json_to_csv* Script

This script converts a `*.json` file into a `*.csv`. It is used to preprocess the data to be understandable by the *weather* package.

```
import requests
from datetime import datetime
import json
from time import sleep
from os import walk
import csv

def list_files_in_folder(dir):
    f = []
    for (dirpath, dirnames, filenames) in walk(dir):
        for file in filenames:
            if file.endswith(".json"):
                f.append(file)
    f.sort()
    return f


def get_total_new_entries(files, dirname, old_entries):
    to_append = []
    last_date = old_entries[-1]['fint']
    for f in files:
        directory = dirname + "/" + f
        with open(directory, 'r') as fi:
            data = json.load(fi)

        new_entries = get_new_entries(last_date, data)
        if (data[-1]['fint']>last_date):
            last_date = data[-1]['fint']
        to_append.append(new_entries)

    return to_append

def read_file(filename):
    with open(filename, 'r') as f:
        data = json.load(f)
    return data


def convert_to_datetime(date_string):
    return datetime.strptime(date_string.split('.')[0], "%Y-%m-%dT%H:%M:%S")


def convert_to_custom_date_format(date):
    return date.strftime("%Y%m%d%H")

```

```
46
47 def get_new_entries(last_date, new_data):
48     to_append = []
49     last_entry_date = convert_to_datetime(last_date)
50     for new in new_data:
51         date = convert_to_datetime(new['fint'])
52         if date > last_entry_date:
53             to_append.append(new)
54     print(f'Found {len(to_append)} new entries')
55     return to_append
56
57
58 def append_new_data_to_json_file(old_entries, new_entries):
59     all_data = old_entries
60     for new in new_entries:
61         all_data = all_data + new
62     save_to_json_file(all_data)
63
64
65 def save_to_json_file(data):
66     filename = 'all_data.json'
67     with open(filename, 'w') as f:
68         json.dump(data, f, indent=4)
69     print(f'Saved to {filename}')
70
71
72 def export_csv(input_file, cols):
73     data = read_file(input_file)
74     output_file = input_file.replace('.json', '.csv')
75
76     with open(output_file, 'w') as f:
77         writer = csv.DictWriter(f, fieldnames=cols, delimiter=';',
    extrasaction='ignore')
78         writer.writeheader()
79         for row in data:
80             x = convert_to_datetime(row['fint'])
81             row['fint'] = convert_to_custom_date_format(x)
82             writer.writerow(row)
83
84     print(f'Saved to {output_file}')
85
86
87 def main():
88     dirname = "jsons"
89     output_file = "all_data.json"
90     cols = ['fint', 'tamin', 'ta', 'tamax', 'ts', 'inso']
91
92     files_to_read = list_files_in_folder(dirname)
93     old_entries = read_file(output_file)
94     new_entries = get_total_new_entries(files_to_read, dirname, old_entries)
95     append_new_data_to_json_file(old_entries, new_entries)
```

```
96      export_csv(output_file, cols)
97
98
99  if __name__ == '__main__':
100     main()
```

Listing B.5: JSON to CSV Script

Universitat Politècnica de Catalunya
Berlin, Juny 2020